UNIVERSITE DE NANTES

ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATERIAUX

Année : 2002

Thèse de Doctorat de l'Université de Nantes

Spécialité : Automatique et Informatique Appliquée

Présentée et soutenue publiquement par

Erwan BRETON

le 24 juin 2002
à l'Université de Nantes

Contribution à la représentation de processus par des techniques de méta-modélisation

Jury

Président	:	M. BOUZEGHOUB	Professeur, Université de Versailles
Rapporteurs	:	J.-M. GEIB	Professeur, Université de Lille 1
		J.-M. JEZEQUEL	Professeur, IRISA, Rennes
Examinateurs	:	N. MOUADDIB	Professeur, Université de Nantes
		R. THORAVAL	Maître de Conférences, Université de Nantes
Invité	:	Y. LENNON	Directeur NT Migration, groupe Sodifrance

Directeur de Thèse : Jean BEZIVIN
Laboratoire : Centre de Recherche en Gestion de Nantes-Atlantique, Université de Nantes

N° ED 0366-066

Remerciements

Je tiens à dire ma plus sincère gratitude à Jean Bézivin pour m'avoir encadré et assisté durant cette thèse. Ses conseils m'ont été d'une grande aide. Son exigence m'a beaucoup appris.

Je remercie Yves Lennon, directeur de la division NT-Migration du groupe Sodifrance qui m'a accueilli pendant ces quatre années, ainsi que Jean-Paul Bouchet, qui a encadré le volet industriel de cette thèse. Ils ont fait montre d'un intérêt constant pour ces travaux, n'hésitant pas à promouvoir leur mise en œuvre sur des projets.

Je remercie Jean-Marc Jezequel et Jean-Marc Geib de m'avoir fait l'honneur d'être mes rapporteurs. Je remercie également Mokrane Bouzeghoub, Noureddine Mouaddib et René Thoraval, qui ont accepté de faire partie de mon jury.

Un grand merci à l'ensemble des personnes qui font ou ont fait partie de l'équipe Process à Sodifrance : Arnaud, Christophe, Laurent, Morgan et Nicolas. Leur contribution aux diverses réalisations industrielles a été extrêmement précieuse. La bonne humeur régnant au sein de l'équipe n'est pas étrangère au plaisir que j'ai pris à travailler avec eux. Je ne saurai oublier Frédéric, Hervé, Jean-Michel, Richard et Sébastien, ainsi que mes autres collègues de NT-Migration, avec qui j'ai eu l'occasion de collaborer au cours de ces travaux.

Enfin, merci à Cristelle, Mewen et Luan. Ils ont supporté mes petites angoisses et mes remises en question. Ils m'ont surtout apporté de grandes joies.

Résumé

Le concept de processus a pris une dimension importante dans le domaine du génie logiciel. La complexité croissante des systèmes d'informations, la rapidité des évolutions technologiques et les nouveaux modes de travail (externalisation et sous-traitance) sont autant d'éléments expliquant ce phénomène. La maîtrise des processus devient donc un enjeu majeur pour les entreprises. C'est dans ce cadre que se situe notre travail. La société Sodifance, partenaire industriel de cette thèse, est spécialisée dans la gestion et l'évolution des systèmes d'information. Au fur et à mesure des projets, elle a acquis un savoir-faire sur les processus de maintenance et de migration. C'est pour collecter et organiser ces connaissances que nous avons proposé un formalisme adapté à la description de ce type de processus, et défini à l'aide de techniques de méta-modélisation. Celles-ci ont radicalement évolué ces dernières années avec l'adoption par l'OMG du MOF. La dernière avancée en date, le MDA, toujours à mettre au crédit de l'OMG, propose une nouvelle approche du génie logiciel basée sur les modèles. A partir de ces spécifications, et des produits de transformation de modèles et de génération de code de la société Sodifrance, nous avons conçu un ensemble d'outils pour la définition et la manipulation de modèles de processus. En particulier, nous avons développé des mécanismes pour l'opérationnalisation des modèles de processus qui ont été validés dans le cadre d'un projet de tierce maintenance applicative mené par Sodifrance. Ce travail a initié un certain nombre de réflexions sur les apports de la méta-modélisation pour la représentation de processus. Nous nous sommes intéressés à l'organisation des méta-modèles de processus et à leur relations avec des méta-modèles dédié à des domaines différents. Enfin, nous avons réalisé un certain nombre d'expérimentations sur la prise en compte des aspects dynamiques des processus et l'intégration des règles spécifiant leur exécution.

Summary

The concept of process is taking a significant place in the field of software engineering. The increasing complexity of information systems, the speed of technological evolution and the new working methods (outsourcing and subcontracting) are some reasons that explain this phenomenon. Process management is thus becoming a major stake for most companies. Our work is situated within this framework. The Sodifrance Company, the industrial partner of this thesis, is specialized in the management and the evolution of information systems. Know-how on the processes of maintenance and migration were acquired as projects were carried out. In order to collect and organize this knowledge we propose a formalism dedicated to the description of this type of processes. It has been defined using meta-modeling techniques. Those radically evolved these last years since the adoption of the MOF by the OMG. This trend leads into the MDA, another proposal of the OMG, which proposes a new approach to software engineering based on models. From these specifications, and the products developed at Sodifrance for model transformation and code generation, we designed a set of tools for the definition and the handling of process models. In particular, we developed mechanisms for operationalizing process models that were validated within an industrial project of applicative third party maintenance managed by Sodifrance. This work initiated some thoughts on the contributions of meta-modeling to the representation of processes. We mainly focused on the organisation of process meta-models and their relations with meta-models dedicated to different fields. We also carried out some experiments on making explicit the dynamic aspects of processes and the rules specifying their execution.

Table des matières

1. Introduction générale

1.1. Le contexte de la thèse

Cette thèse a fait l'objet d'une convention CIFRE (Conventions industrielles de formation par la recherche) entre la société Sodifrance, société de services spécialisée dans la maîtrise et l'évolution des systèmes d'information et le laboratoire CRGNA (Centre de Recherche en Gestion de Nantes Atlantique). Un sujet d'intérêt commun à ces deux entités concerne les techniques de modélisation et de méta-modélisation. La thèse de Richard Lemesle, également réalisée dans le cadre d'un partenariat entre Sodifrance et le CRGNA, avait abouti à la formalisation des sNets, un système de méta-modélisation. A partir de ce formalisme, un certain nombre d'outils ont pu être développés pour la rétro-ingénierie, la production de modèles à partir de programmes, et la manipulation de modèles. Ces outils permettent d'assister, voire d'automatiser, la plupart des tâches constituant les processus métiers de Sodifrance : migration, maintenance d'applications, développement.

Toutefois, l'industrialisation des processus restait partielle. Il manquait des outils pour la gestion des processus eux-mêmes. A partir des différents projets que Sodifrance avait menés, un certain nombre de connaissances et de savoir-faire avaient été acquis. En particulier, des processus (ou des portions de processus) standards avaient été identifiés. Ces processus étaient décrits dans un format textuel, donc de façon informelle et parfois imprécise. De telles spécifications sont difficilement manipulables. Or, dans les domaines d'activités de Sodifrance, les processus doivent souvent être adaptés selon le contexte ou les habitudes de travail du client. De plus, une définition textuelle de processus n'est guère exploitable par des outils logiciels. Il n'était donc pas possible d'utiliser ces définitions de processus pour les planifier, contrôler leur exécution, mesurer leur efficacité et, le cas échéant, les améliorer. Bien sûr, des outils de gestion de processus (outils de planification, base de données de suivi) étaient utilisés lors des différents projets, mais le processus devait être redéfini pour chacun d'entre eux, ce qui entraînait une importante perte de temps, des risques d'incohérences, ainsi que des difficultés d'adaptation. Il manquait donc à Sodifrance des mécanismes pour définir des processus adaptés à son domaine d'application et pour exploiter ces définitions.

C'est ce constat qui a amené Sodifrance et le CRGNA à définir un travail de thèse sur le thème des processus. Sodifrance ayant intégré les concepts de modélisation et de méta-modélisation dans sa culture d'entreprise, il nous a semblé naturel de baser nos travaux sur ce type de techniques. Ceux-ci avaient donc pour but de proposer un formalisme de définition de processus, un méta-modèle, adapté aux domaines d'application de Sodifrance, et tenant compte du vocabulaire et des usages développés par les experts métier. Il ne

s'agissait pas de forcer ces derniers à se plier à une nouvel ensemble de concepts, mais plutôt de fournir un support explicite à celui en vigueur. De plus, une fois ce formalisme défini, il fallait également étudier comment une définition du processus pouvait être exploitée pour différents besoins : planification, assistance, contrôle et suivi de l'exécution, etc. Ce document présente le résultat d'un certain nombre de ces travaux, parmi les plus significatifs, ainsi que les réflexions qu'ils ont pu engendrer.

1.2. Les processus

Les processus font aujourd'hui l'objet d'une attention toute particulière, et ce quel que soit le domaine. Un nombre important de propositions mettant en avant ce concept ont récemment vu le jour. WSFL (Web Service Flow Language), développé par la société IBM et publié en 2001, et XLANG, mis au point par la société Microsoft et également publié en 2001, proposent des formalismes basés sur XML pour la définition de processus à base de services Web. ebXML (Electronic Business using XML), tout comme BPML (Business Process Modeling Language), eux aussi publiés en 2001, définissent des dialectes XML pour la spécification des processus d'affaire collaboratifs. A l'OMG (Object Management Group), les travaux sur EDOC (Enterprise Distributed Object Computing) sont en cours de finalisation. Cette spécification vise à la description des composants d'un système, et également des processus qui les mettent en œuvre. Enfin, SPEM (Software Process Engineering Metamodel) est une autre spécification de l'OMG également en cours de finalisation, plus particulièrement dédiée à la définition des processus de développement logiciel.

Ces multiples exemples sont significatifs de l'intérêt que porte aujourd'hui le monde industriel à la notion de processus. Pour finir de s'en convaincre, il suffit de taper la requête « process model » sur le moteur de recherches Google. Le nombre de résultats renvoyés se monte à près de 500000. Même si toutes ces références ne sont pas pertinentes, ce nombre nous paraît significatif. Et cette tendance peut être observée dans différents contextes. Dans le domaine du contrôle de gestion, la méthode ABC (Activity-Based Costing) recommande l'approche par les activités, et non plus par les produits comme le font les méthodes traditionnelles. Dans la dernière version de la norme ISO 9000, qui concerne le domaine de la gestion de la qualité, le consortium ISO (International Organization for Standardization) a également intégré une approche processus. On assiste à une profusion de nouveaux concepts, outils et formalismes en rapport avec la notion de processus.

La maîtrise des processus devient donc un enjeu majeur pour les entreprises, et cela pour plusieurs raisons. Tout d'abord elle peut permettre une amélioration des performances que ce soit en coût, en délai ou en qualité. En effet, toute amélioration a un préalable, l'identification de la cause du dysfonctionnement. L'explicitation des différentes activités constituant le processus et de leurs relations de dépendance fournissent des moyens d'analyse pour déterminer où se situent le problème et ses origines. De plus, à l'heure où la plupart des tâches individuelles sont plus ou moins assistées par ordinateur, un gain de performance important réside dans l'optimisation de leur synchronisation. La maîtrise des processus est également un garant d'une plus grande adaptabilité. Adaptabilité à la

demande, à la concurrence, mais également à l'évolution de la législation (par exemple, la loi sur la réduction du temps de travail adoptée en France) ou aux nouvelles technologies. Dans ce dernier cas, l'Internet et les autres technologies similaires ont multiplié le nombre de canaux de communication potentiels. Ainsi, une relation entre un client et sa banque peut se faire à travers différents médias (téléphone, courrier, mail, Internet), chacun d'entre eux impliquant des contraintes différentes en terme de sécurité, ce qui entraîne fréquemment l'adaptations des processus standards.

Cette maîtrise des processus peut être réalisée par différents moyens. A un niveau rudimentaire, un référentiel documentaire informe les acteurs de l'entreprise sur la façon de mettre en œuvre les processus. Afin de les assister, un certain nombre de composants logiciels implémentant tout ou partie du processus ont pu être développés. Enfin, depuis ces dernières années, on voit apparaître des produits, tels que les systèmes de gestion de workflow ou d'EAI (Enterprise Application Integration), qui permettent d'automatiser le contrôle d'exécution des processus, en prenant en compte ces derniers de façon explicite. La distinction communément faite entre le workflow et l'EAI est que le premier est plutôt dédié à l'exécution de processus organisationnels, alors que l'EAI se concentre sur les processus totalement automatisés. La frontière entre ces deux mondes tend aujourd'hui à disparaître. On assiste à une intégration de ces deux notions au sein d'une même offre chez de nombreux industriels (Vitria, Tibco).

Que ce soit à des fins d'information, d'analyse, de simulation ou d'automatisation, l'exploitation d'un processus implique que ce dernier soit décrit. Cette description peut se faire en langue naturelle mais une telle représentation n'est guère exploitable. Des formalismes spécialement dédiés existent également. Nous en avons déjà cité un certain nombre : ebXML, EDOC, SPEM, WSFL ou XLANG. Il en existe bien d'autres, tant dans le milieu universitaire que dans le monde industriel. Chacun d'eux est adapté à un contexte bien particulier. Cette spécialisation peut être observée au niveau du contenu du formalisme, des concepts qu'il définit, mais également au niveau de ces caractéristiques (lisibilité, exécutabilité, extensibilité, etc.). Une première constatation qui s'impose est donc qu'il ne peut y avoir un seul et unique formalisme de processus, adapté à tous les contextes. Le formalisme utilisé pour l'analyse d'un processus n'est pas forcément le même que celui utilisé pour son automatisation. Le contrôleur de gestion, le qualiticien, l'organisateur et l'ingénieur en informatique n'ont pas les mêmes points de vue. Ils ont donc des besoins d'expression différents. La constatation suivante est alors qu'il est nécessaire d'établir des ponts entre ces formalismes, pour faciliter le passage d'un point de vue sur le processus à un autre. C'est pourquoi on voit se nouer des partenariats entre des sociétés éditrices d'outils de définition et d'analyse de processus et des sociétés éditrices de solutions d'exécution de processus. Ainsi, MEGA et Akazi, IDS Scheer et Staffware, proposent des mécanismes permettant d'établir des couplages entre leurs produits respectifs.

1.3. Les nouvelles techniques de méta-modélisation

Le paysage de la méta-modélisation a radicalement évolué au cours de ces dernières années. La plupart de ces évolutions sont à porter au crédit de l'OMG. Le MDA [92] (Model Driven Approach), nouvelle approche du génie logiciel, marque à la fois un aboutissement et un début. Un aboutissement, car il intègre l'ensemble des dernières spécifications adoptées par l'OMG dans une architecture organisée. Un début, car beaucoup de questions sur sa mise en pratique effective restent encore en suspens.

L'OMG s'est longtemps concentré sur l'interopérabilité entre composants logiciels exécutables. CORBA (Common Object Request Broker Architecture) s'inscrit totalement dans cette problématique. CORBA définit les spécifications d'un middleware permettant de faire interagir des programmes hétérogènes. En plus de CORBA, l'OMG a également développé le langage IDL (Interface Definition Language) dont l'objectif est de définir des interfaces de manière normalisée. IDL est accompagné d'un certains nombre de spécifications de mises en correspondance (mappings) entre ce langage pivot et les principaux langages de programmation (C++, Java, Cobol, etc). Ces spécifications ont été concurrencées par d'autres propositions propriétaires et donc moins ouvertes car elles contraignent à l'utilisation d'un langage ou d'une plate-forme unique (Java/RMI, EJB, COM+). Ces différentes alternatives se bornent toutefois à l'interopérabilité des objets à l'exécution.

L'adoption d'UML (Unified Modeling Language) en 1997 a représenté un tournant dans la stratégie de l'OMG. Avec UML, on est passé du niveau des composants implémentés, où plusieurs langages différents coexistent, à celui de la modélisation, où chaque composant peut être exprimé en UML. L'intérêt d'UML est qu'il permet d'introduire bien plus d'information que les interfaces IDL. Ainsi, on peut définir au niveau d'UML des contraintes explicites via le langage OCL (Object Constraint Language), ainsi qu'un certain nombre d'aspects comportementaux (diagrammes d'état, de séquence, de collaboration). Toutefois, le périmètre d'UML se cantonne à la représentation des artefacts manipulés lors du développement de logiciels objet. En particulier, il ne prend pas en compte la définition du processus logiciel. Il n'est donc qu'un méta-modèle parmi beaucoup d'autres.

La crainte de voir se multiplier des méta-modèles isolés a poussé l'OMG à adopter le MOF en 1997. Le MOF est un méta-méta-modèle unique et réflexif, fournissant un langage de base pour la définition de méta-modèles. On a donc assisté, avec le MOF, à l'apparition d'un second bus d'interopérabilité. Alors que CORBA permet de faire interagir des composants à l'exécution, le MOF permet d'établir des relations entre modèles. Le MOF se situe donc à un niveau d'abstraction supérieur à celui de CORBA. Cela permet de pouvoir décrire les liaisons entre systèmes hétérogènes dès les phases d'analyse ou de conception, au lieu de limiter leur définition au système implémenté. Ces deux propositions ne sont pas complètement disjointes puisque des interfaces IDL ont été définies pour la manipulation des modèles basés sur des méta-modèles MOF.

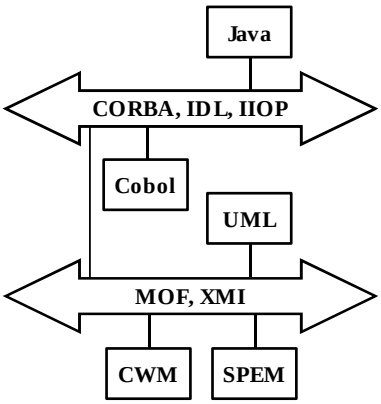


Figure 1. Les deux bus d'interopérabilité de l'OMG

Avec l'émergence de la notion de méta-modèle, on assiste à une transition technologique des programmes à objet vers les modèles, de l'OMA (Object Management Architecture) vers le MDA. Ce dernier se base sur le MOF et UML, ainsi que sur XMI (XML Metadata Interchange), spécification pour la sérialisation et l'échange de modèles et de méta-modèles en XML, et sur CWM (Common Warehouse Metamodel), méta-modèle qui définit un certain nombre de structures de données (base de données relationnelle, Data Division Cobol, schéma XML, etc.). UML permet de construire, de voir, et de manipuler les modèles. Les référentiels MOF fournissent les services de stockage et XMI ceux d'échange d'information. Enfin, les systèmes existants peuvent être intégrés par l'intermédiaire de CWM.

Un des objectifs majeurs du MDA est d'assurer la stabilité du système d'information face aux évolutions technologiques. Celle-ci est assurée par la séparation des informations métier des aspects spécifiques à la plate-forme d'exécution au travers des PIM (Platform-Independent Model) et des PSM (Platform-Specific Model). Un PIM est un modèle défini en utilisant UML ou un autre méta-modèle, qui se veut totalement indépendant de toute plate-forme d'exécution, c'est-à-dire qu'il ne contient que des informations relatives aux métiers de l'entreprise. Un PSM est la projection d'un PIM vers une plate-forme particulière. Un PSM peut être défini en utilisant un profil UML spécifique, tel que le profil UML pour CORBA [71]. Il peut également être défini sous la forme d'interfaces dans la plate-forme cible (par exemple un ensemble d'interfaces IDL si CORBA est la plate-forme d'exécution). CORBA n'est ici considéré que comme une plate-forme d'exécution parmi d'autres (J2EE, .NET, etc.). Le but avoué est de découpler totalement le modèle initial, basé sur les invariants du domaine considéré, de sa conversion vers une cible d'exécution particulière.

La transition d'un PIM vers un PSM se veut au moins partiellement automatisée. Les techniques de transformation de modèles et de génération de code vont donc occuper une place non négligeable au sein du MDA [3]. La part de code généré doit augmenter au fur et à mesure que le MDA gagnera en maturité et que les modèles seront plus complets et précis (voir la partie sur Action Semantics). Une évolution majeure concerne donc le statut des modèles dans le processus de développement logiciel. Du document plus ou moins précis guidant l'informaticien dans ses développements, ils deviennent des supports d'information réutilisables, et ils peuvent être exploités pour produire du code de façon automatisée. Le code d'implémentation n'est alors plus qu'un produit dérivé du modèle.

L'architecture globale ainsi que le formalisme (le méta-méta-modèle) étant stabilisés, les initiatives se sont déplacées de la standardisation du MOF à la normalisation de méta-modèles par domaines d'activité. Après la spécification de mécanismes horizontaux (communs à tous les méta-modèles), tels que l'extension, les formats de communication, etc, on assiste aujourd'hui à la définition de services verticaux, c'est-à-dire à la proposition de formalismes dédiés, que ce soit sous la forme de méta-modèles ou de profils UML. Un profil UML est un ensemble d'extensions du noyau de base d'UML, lui ajoutant de nouvelles entités, de nouveaux attributs et des contraintes supplémentaires. Parmi ces travaux, nous avons déjà cité SPEM et EDOC. On peut ajouter à cette liste :

- La définition d'un profil UML pour l'EAI [70]
- La définition d'un profil UML pour la définition de processus workflow [69]
- La définition d'un profil UML adapté à l'ingénierie système
- Etc.

On se retrouve donc face à un nombre important de méta-modèles, chacun dédié à un domaine bien spécifique. Parmi ceux-ci, plusieurs intègrent la notion de processus. SPEM, EDOC, et même UML avec les graphes d'activités, peuvent être utilisés pour décrire des processus. Chacun de ces méta-modèle définit toutefois son propre vocabulaire, ses propres concepts et ses propres mécanismes. Dans l'optique où l'utilisation des techniques de méta-modélisation se répande et dépasse du cadre du seul génie logiciel, le nombre de méta-modèles pouvant redéfinir de tels mécanismes serait alors singulièrement accru. Ainsi, des méta-modèles destinés à l'ABC ou à la norme ISO 9000 pourraient alors apparaître, sans que ceux-ci n'aient vocation à assister la production de composants logiciels.

1.4. Objectifs de la thèse

1.4.1. Pertinence des techniques de méta-modélisation pour la description de processus

Un formalisme de description de processus peut prendre des formes extrêmement variées. Comme nous le verrons par la suite, il peut être défini comme un langage, une ontologie ou encore un dialecte XML. La méta-modélisation n'est qu'un outil supplémentaire à notre disposition. Toutefois, nous avons fait le choix de baser l'ensemble de nos travaux sur ces techniques. Il s'agit en partie d'un choix d'opportunité. Les récentes propositions de l'OMG (MOF, UML, XMI et MDA) nous poussaient dans cette direction. Les travaux du CRGNA autant que la culture d'entreprise de Sodifrance nous confortaient également dans ce choix. Les différentes réalisations menées autour des sNets, de la transformation de modèles et de la génération de code à partir de modèles sont autant de briques de base sur lesquelles s'appuient nos propres travaux. De plus, nous avons la conviction que l'application des techniques de méta-modélisation à la description de processus offre des perspectives extrêmement intéressantes. Nous nous efforçons donc, tout au long de ce document, de faire ressortir différents bénéfices que nous avons pu identifier.

1.4.2. Caractérisation des méta-modèles de processus

Dans une même entreprise, le qualiticien, le contrôleur de gestion, l'organisateur, le chef de projet et l'informaticien ont chacun leur propre point de vue sur les processus. A chacun son domaine de compétence particulier, et donc à chacun son ou ses méta-modèles spécifiques. Le contrôleur de gestion sera plus intéressé par les notions de coûts et de performance, le qualiticien par la définition des procédures, le chef de projet par l'affectation des ressources et la planification et l'informaticien par l'automatisation et la connexion avec des composants logiciels. Ces points de vue ne sont bien sûr pas complètement disjoints, mais ils représentent différents aspects d'un même système.

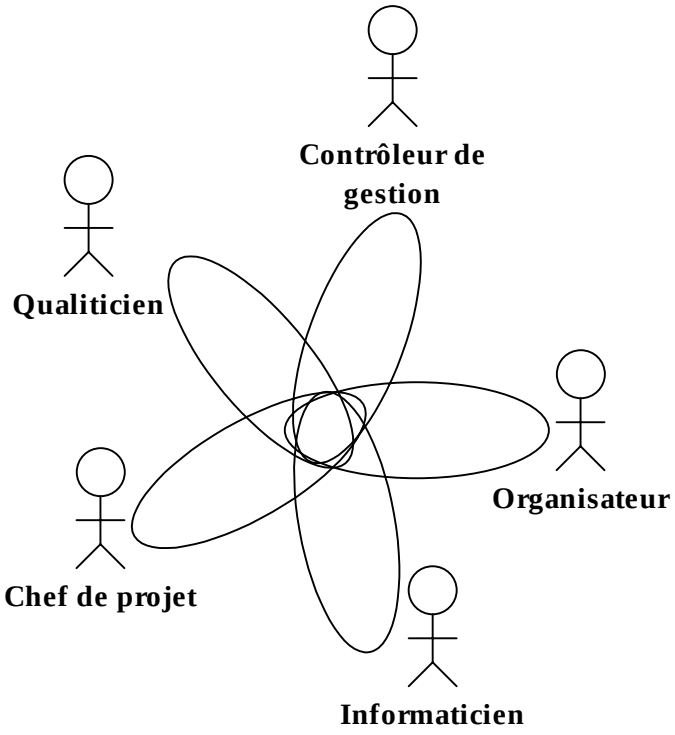


Figure 2. Différents points de vue sur les processus

Tous ces types de processus présentent bien des spécificités. Toutefois il nous semble qu'ils partagent aussi de nombreux points communs. Un des objectifs de cette thèse est donc de comparer un certain nombre de standards issus de ces différents domaines, qu'ils soient de facto ou de jure, afin de déterminer si on peut déterminer un noyau commun à tous. Cette étude est rendue aujourd'hui possible par les nouvelles avancées dans le domaine de la méta-modélisation. Nous pouvons en effet décrire les différentes ontologies que nous avons choisies d'étudier de façon uniforme, ce qui nous permet de faire apparaître plus facilement leurs similarités et leurs différences.

1.4.3. Couplage avec d'autres méta-modèles

Pour être utilisable dans un contexte donné, la définition d'un processus ne peut faire l'économie de la prise en compte de l'environnement d'exécution. Il faut préciser les règles d'affectation de chacune des tâches entre les différentes entités de l'entreprise. Les règles métier conditionnant les transitions entre étapes doivent être définies en fonction des

données métier. Il faut également déterminer les applications pouvant être invoquées et les paramètres qui doivent leur être passés. En ce sens les processus peuvent être considérés comme de véritables vecteurs d'intégration des différentes dimensions de l'entreprise (cf Figure 3). Cette utilisation du processus est d'ailleurs souvent mise en avant dans le monde industriel. De nombreux éditeurs de moteur de workflow insistent sur les apports de leur produit en terme d'intégration. De l'autre côté les outils d'EAI utilisent le terme de « business process » pour nommer les chaînes d'appel aux applications de l'entreprise.

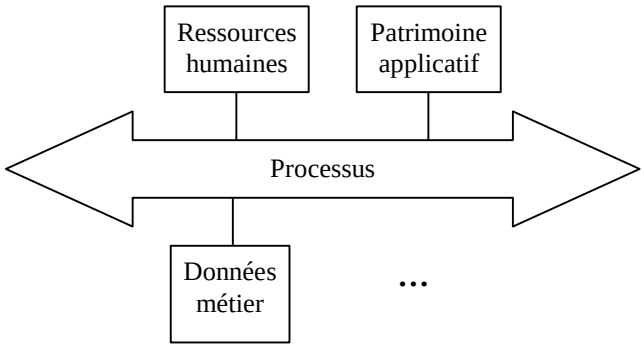


Figure 3. Les processus vus comme des vecteurs d'intégration

Un objectif de cette thèse est donc d'étudier ce que les nouvelles techniques de modélisation peuvent apporter dans l'intégration des processus. En effet, si l'on considère que chaque dimension de l'entreprise est représentée par un méta-modèle indépendant, il est nécessaire de préciser leurs interactions respectives. Un méta-modèle de processus pourra être ainsi couplé avec un méta-modèle de produit, un méta-modèle de la structure organisationnelle, etc. La définition formelle de ces liens entre espaces à priori totalement indépendants est possible car tous les méta-modèles se basent sur un unique méta-méta-modèle. Ils sont donc exprimés dans le même formalisme. Le nombre de méta-modèles pouvant ainsi être mis en relation peut être très important. Comme on peut avoir plusieurs méta-modèles en concurrence pour un même domaine, on voit se dessiner un treillis extrêmement dense de méta-modèles. Afin d'éviter de nous disperser nous insisterons principalement sur le couplage entre le produit et le processus. Les liens existant entre ces deux espaces ont souvent été mis en évidence, et ce bien avant l'apparition des méta-modèles explicites.

1.4.4. Opérationnalisation des modèles de processus

Un modèle de processus est le reflet d'un ensemble de processus réels qui ont été, sont ou seront exécutés. Dans de nombreux cas on souhaite pouvoir exécuter le modèle, que ce soit pour simulation ou pour guider le travail effectif des acteurs. Il est donc nécessaire de spécifier les aspects dynamiques du méta-modèle de processus, c'est-à-dire les concepts et les règles représentant et contraignant l'exécution d'un processus. Ceci est d'autant plus vrai que l'on souhaite automatiser le contrôle de l'exécution des processus. Même dans le cas d'une exécution « manuelle » du processus peuvent apparaître des différences d'interprétation. Par exemple imaginons un méta-modèle de processus définissant le concept de temporisation. Une temporisation est en quelque sorte une transition entre deux étapes, une cible et une source, qui définit un temps T. Lorsque l'étape source a été activée depuis un temps supérieur à T, l'étape cible est activée. Mais qu'en est-

il de l'étape source? Est-elle encore active (la temporisation étant alors vue comme un simple événement)? Est-elle stoppée (la temporisation étant alors vue comme une exception similaire à un « timeout »)? Bien sûr, une documentation bien conçue pourrait combler cette lacune. Toutefois une expression plus formelle ou du moins explicite et précise du comportement serait intéressante. Tout d'abord cela peut être utile pour valider le méta-modèle ainsi que les modèles qui en découlent et qui pourraient être simulés. Ensuite les comparaisons entre méta-modèles de processus ne se limiteraient plus aux simples aspects descriptifs, mais pourraient également prendre en compte le comportement à l'exécution des différentes entités du méta-modèle. Enfin ces règles pourraient servir de base à l'implémentation de moteurs d'exécution. Notre objectif n'est pas ici de développer un nouveau langage mais d'étudier la réutilisabilité de travaux déjà menés dans le cadre de la sémantique des langages de programmation ou sur la sémantique des actions dans UML.

1.4.5. Instrumentation des processus métier de Sodifrance

Enfin, un dernier objectif de cette thèse, et non le moindre, concerne l'instrumentation des processus métier de Sodifrance (maintenance applicative, migration, développement logiciel). Il s'agit de proposer des outils simples et flexibles pour la définition des processus et la production d'un ensemble de fonctionnalités pour le contrôle et le suivi de leur exécution. Cette génération doit se faire de façon la plus souple possible. En effet, les projets de Sodifrance étant menés dans des environnements très différents les uns des autres, le même modèle de processus doit pouvoir être adapté dans des contextes extrêmement variés. C'est dans ce cadre que nous avons développé un outillage autour de la définition et de la manipulation de modèles de processus. Cet outillage a pu être testé sur un certain nombre de prototypes. Il a également été utilisé pour déployer un processus automatisé sur un projet de tierce maintenance applicative. Ces différents travaux nous ont permis de l'éprouver et de déterminer un certain nombre de pratiques.

1.5. Organisation du document

L'organisation de ce document découle directement de la façon dont s'est déroulée cette thèse. La première partie a consisté à appliquer les dernières techniques de méta-modélisation dans le cadre de projets industriels. Ces réalisations ont soulevé de nombreuses questions. Nous en avons résolu certaines lors des projets, quelquefois de façon ad hoc, tandis que d'autres restaient en suspens. Dans un deuxième temps nous avons tenté de leur apporter des réponses génériques. Nous avons également étendu notre champ d'investigation à des problématiques plus générales qui nous ont semblé importantes.

Pour respecter cette logique, le présent document est organisé en trois grande parties. Dans la première (chapitres 2 à 5) nous explicitons l'ensemble des concepts que nous serons appelés à manipuler. Dans la suivante (chapitres 6 à 8) nous présentons quelques réalisations industrielles significatives. Enfin, dans la dernière partie (chapitres 9 à 12) nous livrons un certain nombre de réflexions issues de ce qui a été fait (ou ce qui aurait pu être

fait) dans le cadre des applications industrielles, et basées sur les concepts que nous avons présentés dans la première partie.

Le chapitre 2 de ce document résume les récentes avancées qui ont eu lieu dans le domaine de la méta-modélisation. Nous y traitons des nouveaux standards apparus à l'OMG (MOF et XMI) et de travaux similaires, les sNets, résultat d'une collaboration préalable entre le CRGNA et Sodifrance. Nous y introduisons également des techniques dédiées à la transformation de modèles comme XSLT, ainsi que des outils propriétaires développés par Sodifrance : Scriptor-Generation et Scriptor-Transformation. Enfin, nous terminons en étudiant les enjeux du MDA et les problèmes qu'il soulève.

Le chapitre 3 traite de la notion de processus dans différents contextes. Nous étudions l'utilisation qui est faite des processus dans les domaines de l'ingénierie système, du contrôle de gestion et de l'organisation, de la gestion de la qualité et du génie logiciel. Nous tentons de faire apparaître les concepts sous-jacents à celui de processus à travers les définitions qui en sont donné, ainsi que les bénéfices liés à cette approche.

Dans le chapitre 4, nous présentons plusieurs formalismes de représentation de processus sous la forme de méta-modèles. Cette représentation uniforme permet une compréhension plus aisée de chacun d'entre eux. Elle facilite également leur comparaison en faisant apparaître les structures communes et les particularités de chacun.

Le chapitre 5 traite de l'exécution des modèles. Nous commençons par introduire la notion d'exécution de processus telle qu'elle est définie dans le domaine du workflow. Nous présentons ensuite, à titre d'exemple, deux algèbres de processus, celle de Tony Hoare et celle de Robin Milner. Leurs modèles, respectivement CSP et CCS, formalisent la représentation des processus et leur exécution à l'aide de concepts mathématiques. Enfin, nous traitons d'autres techniques de spécification de la sémantique d'exécution. Nous nous intéressons tout particulièrement à la sémantique des actions, à travers deux propositions différentes, la première destinée aux langages de programmation, et la seconde aux modèles UML.

Dans le chapitre 6 nous présentons l'outillage développé au sein de la société Sodifrance pour la définition de modèles de processus et leur manipulation. Nous commençons par étudier le méta-modèle de processus que nous avons proposé pour la description des processus métier de Sodifrance. Puis nous introduisons le modeleur graphique de processus basé sur ce méta-modèle, ainsi que les mécanismes d'ouverture permettant la reprise de modèles basés sur d'autres formalismes et l'opérationnalisation des modèles de processus.

Le chapitre 7 relate une mise en œuvre de cet outillage sur un cas concret : un processus de tierce maintenance applicative. Nous présentons ce processus tel que nous l'avons modélisé, c'est-à-dire conformément au formalisme précédemment défini, à l'aide du modeleur graphique. Ce modèle a ensuite été exploité pour alimenter un moteur de workflow et ainsi automatiser le contrôle de l'exécution du processus. Nous insistons particulièrement sur la méthode que nous avons mise en pratique, une approche itérative par prototypage, permettant une validation rapide et sélective. Les techniques de génération

nous permettent en effet de produire instantanément des prototypes opérationnels à partir de modèles partiels. Ces prototypes peuvent alors servir comme base de communication avec les experts métier, et comme support de validation.

Dans le chapitre 8 nous rapportons une expérimentation supplémentaire consistant à intégrer un outil de planification en plus du modeleur de processus et du moteur de workflow. Nous montrons comment nous avons automatisé les échanges d'informations entre les différents modèles impliqués : le modèle de processus générique, le modèle de processus exécutable et le modèle de planification. Pour cela nous utilisons les deux outils de manipulation de modèles développés par Sodifrance, Scriptor-Generation et Scriptor-Transformation. Cette intégration n'est pas limitée à la phase de conception, puisque la planification est maintenue cohérente avec l'exécution réelle du processus par l'intermédiaire de composants de mise à jour intégrés dans le workflow.

Dans le chapitre 9, nous nous intéressons à l'existence d'un ensemble de concepts communs à chaque méta-modèle de processus. Pour cela, nous utilisons les différents méta-modèles que nous avons étudiés dans la première partie. Nous tentons de les comparer en nous basant sur un certain nombre d'aspects qui nous semblent particulièrement pertinents. Nous explicitons ensuite les résultats de cet alignement sous la forme d'un méta-modèle qui pourrait donc constituer un noyau de méta-modèle de processus.

Dans le chapitre 10 nous nous intéressons aux relations entre les méta-modèles de processus et d'autres types de méta-modèles (produits, organisation structurelle). Nous ciblons le couplage existant entre modèle de produit et modèle de processus, couplage constaté dans de multiples propositions. Nous identifions un certain nombre de relations entre UML et le méta-modèle de processus que nous avons présenté dans le chapitre 6.

Le chapitre 11 pose le problème de la représentation de l'exécution de processus. Au travers d'expérimentations avec les réseaux de Petri et des formalismes de définition et de contrôle de workflow, nous traitons de la modélisation d'entités dynamiques. Nous tentons de décrire de façon explicite l'exécution d'un processus, ainsi que ses liens avec la définition de ce même processus. Nous terminons par une comparaison entre notre vision et celle véhiculée par l'architecture de modélisation à quatre couches. Nous insistons tout particulièrement sur les notions d'instanciation et d'exécution.

Dans le chapitre 12, nous présentons une étude théorique sur l'exécutabilité des modèles de processus. Nous commençons par formuler le problème, qui est le manque d'explicitation de la sémantique d'exécution des modèles de processus dans les systèmes de méta-modélisation basés sur le MOF. Nous indiquons ensuite une solution implémentée dans le système des sNets. Celle-ci présentant quelques lacunes, nous proposons une solution alternative basée sur des formalismes existants de sémantique d'action.

Finalement, le chapitre 13 conclut ce travail en récapitulant les résultats obtenus et les principales contributions et en évoquant différentes possibilités de continuation et d'extension du travail déjà réalisé.

Partie I
DEFINITIONS ET CONCEPTS

Introduction

Tout au long de nos travaux, nous nous sommes souvent basés sur des propositions existantes. Les réalisations industrielles que nous avons effectuées utilisent des techniques de méta-modélisation. Nous avons ainsi pu mettre en œuvre le formalisme des sNets, finalisé dans la thèse de Richard Lemesle [49], ainsi que des formalismes standards issus de l'OMG. Ces réalisations, tout comme nos réflexions, ont été abondamment alimentées par les nombreux travaux dans le domaine du processus. Ces derniers ont d'ailleurs été extrêmement nombreux pendant la durée de cette thèse, donnant lieu à de nombreuses propositions importantes (SPEM, EDOC, BPML, XLANG, WSFL, etc.). La rapidité à laquelle a évolué l'état de l'art dans ce domaine, tout en confirmant l'opportunité de cette thèse, a constitué un facteur de complexité supplémentaire.

Dans cette première partie nous présentons les concepts sur lesquels se basent nos travaux. Nous commençons par dresser un panorama des techniques de méta-modélisation. Nous y présentons deux formalismes (le MOF et les sNets) et leurs formats d'échange associés. Nous étudions également quelques techniques de transformation de modèle que nous avons eu l'occasion de mettre en œuvre. Les parties suivantes traitent du concept de processus. Tout d'abord, nous passons en revue l'utilisation qui est faite du processus dans quelques domaines que nous avons pu aborder (ingénierie système, contrôle de gestion et organisation, gestion de la qualité et développement logiciel). Ensuite, nous introduisons différents formalismes de processus. Ces formalismes nous ont servi de base de travail lors de la définition d'un méta-modèle pour les processus métier de Sodifrance. Ils constituent également la clé de voûte d'un travail de réflexion que nous avons mené sur la caractérisation et l'organisation des méta-modèles de processus, et que nous présentons ultérieurement. Enfin, dans la dernière section de cette partie, nous traitons de l'exécutabilité des modèles. Les concepts qui y sont présentés sont largement référencés dans la section présentant nos travaux sur la définition de la sémantique d'exécution des méta-modèles de processus.

2. Les techniques de méta-modélisation

2.1. Introduction à la méta-modélisation

La modélisation est une technique permettant de représenter des systèmes. Les objectifs sont d'aider dans l'analyse des systèmes existants (modélisation « as-is »), dans la conception de nouveaux systèmes (modélisation « to-be »), ainsi que de fournir un support à la discussion et à l'échange d'informations entre les différentes parties intéressées. Un modèle n'est pas, dans la plupart des cas, une représentation exhaustive du système. En effet, il serait alors aussi complexe que le système lui-même. Pour qu'il soit lisible, compréhensible et manipulable, on le restreint à une description partielle du système. Selon les objectifs du modélisateur, les aspects jugés pertinents sont pris en compte, tandis que les autres sont occultés. Ainsi, un modèle correspond à un certain point de vue sur un système. Il est le produit de l'application d'un filtre sur un système. Ce filtre, qui définit les aspects d'intérêt du système, est le méta-modèle. En délimitant l'univers du discours, il contraint la définition d'un modèle à partir d'un système. Un même système peut donc donner lieu à une infinité de modèles ciblant chacun des aspects différents, autrement dit basés sur des méta-modèles différents. C'est ce que montre la Figure 4, extraite de [4]. Il est également fréquent qu'un même méta-modèle permette d'extraire plusieurs modèles selon certains choix arbitraires de représentation.

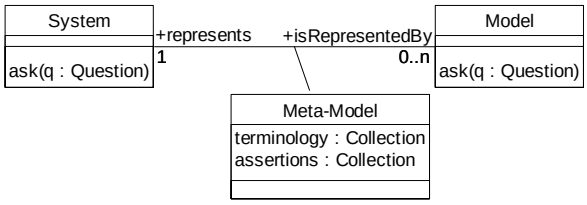


Figure 4. Système, modèle et méta-modèle

Un méta-modèle peut être découpé en deux parties distinctes : la terminologie et les assertions. La terminologie est l'ensemble des concepts à disposition, leurs propriétés et leurs relations. Ces concepts doivent avoir une certaine généricité, car ils doivent pouvoir être réutilisés par plusieurs modèles. C'est principalement ce niveau de généricité qui distingue le méta-modèle du modèle. Les assertions sont des règles supplémentaires permettant de contraindre l'utilisation des éléments introduit par la terminologie (concepts, propriétés et relations).

Etant donné qu'on peut définir un méta-modèle spécifique pour chaque domaine (et dans bien des cas on peut en définir plusieurs pour un même domaine), le risque était grand

de les voir se multiplier de façon totalement anarchique. C'est pour y faire face qu'est apparu le concept de méta-méta-modèle. Un méta-méta-modèle fournit le langage unique de définition de l'ensemble des méta-modèles, ce qui facilite leur interopérabilité. On a alors assisté à l'émergence d'une architecture de modélisation basée sur quatre couches distinctes (Figure 5), autour de laquelle s'est formé un consensus. Au niveau le plus haut (aussi appelé M3) se trouve le seul et unique méta-méta-modèle. Au niveau inférieur (M2) il y a des méta-modèles définissant chacun un formalisme propre à un domaine donné. Puis au niveau M1, on a les modèles, chaque modèle étant basé sur un et un seul méta-modèle. Enfin au niveau le plus bas (M0) se trouvent les « instances » des modèles. Cette architecture de méta-modélisation est aujourd'hui communément acceptée, particulièrement grâce aux travaux de l'OMG en la matière. Des formalismes moins récents comme CDIF [25] (CASE Data Interchange Format) ou IRDS [39] (Information Resource Dictionary System) avaient ouvert la voie.

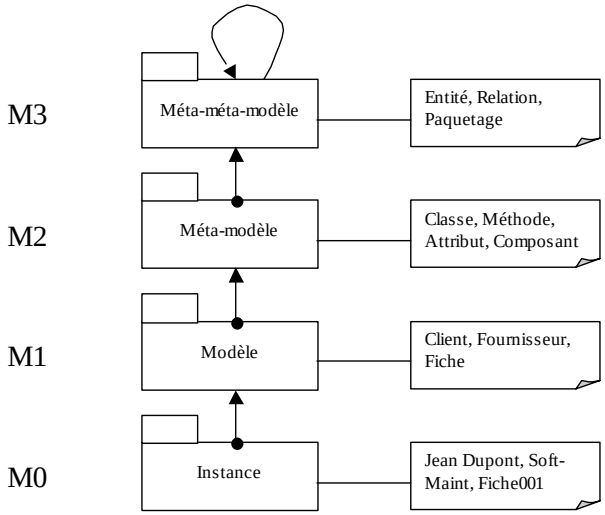


Figure 5. Une illustration de l'architecture 4 couches

Dans ce chapitre nous présentons certaines des techniques qui font partie de la boîte à outils de la méta-modélisation. Tout d'abord nous étudions les méta-méta-modèles et plus particulièrement la spécification de l'OMG, le MOF, et la proposition de l'Université de Nantes, les sNets. Ensuite nous passons en revue les formats d'échange normalisés définis pour chacun des formalismes étudiés précédemment. Ils permettent l'échange de modèles entre composants logiciels. Ce sont XMI et XML-sNets, qui s'appuient tous deux sur XML. Nous avons limité cette étude à l'interopérabilité par échange de fichiers, omettant d'autres mécanismes comme la correspondance MOF IDL. Pour des raisons de concision nous avons choisi de nous limiter aux techniques que nous avons pu mettre en œuvre. Puis nous présentons des outils dédiés à la transformation de modèles. Pour cette partie, nous avons également choisi de ne pas dresser un panorama exhaustif des outils de transformation, mais de restreindre notre présentation à ceux que nous avons effectivement employés. Nous terminons en présentant le MDA, une nouvelle approche du développement logiciel proposée par l'OMG, et qui se base principalement sur les modèles et les méta-modèles.

2.2. Des méta-méta-modèles

2.2.1. Un formalisme normalisé : le MOF

Le MOF a été adopté en 1997 par l'OMG. Il spécifie un langage pour la définition de méta-modèles (c'est donc un méta-méta-modèle) ainsi qu'une architecture de modélisation. Il a pour objectif de faciliter la gestion des méta-données et leur interopérabilité. Le MOF définit un méta-méta-modèle réflexif non minimal. Il est directement issu des travaux sur UML. En effet, à l'origine, UML était réflexif, c'est-à-dire qu'UML était décrit en UML. C'est cette partie réflexive d'UML, légèrement remaniée, qui a donné naissance au MOF. Aujourd'hui ces deux spécifications évoluent indépendamment. Toutefois, les différences entre la version actuelle du MOF (1.4) [63] et le noyau de la version actuelle d'UML (1.4) [67] sont relativement limitées. Le méta-méta-modèle MOF présenté Figure 6 ne diverge avec le noyau d'UML que sur un petit nombre de points. Par exemple, en UML, *GeneralizableElement* n'hérite pas de *Namespace* mais de *ModelElement*, une relation explicite existe entre *Classifier* et *Feature*, alors que dans le MOF cette relation est définie par la relation *Contains* entre un *Namespace* et un *ModelElement*, etc.

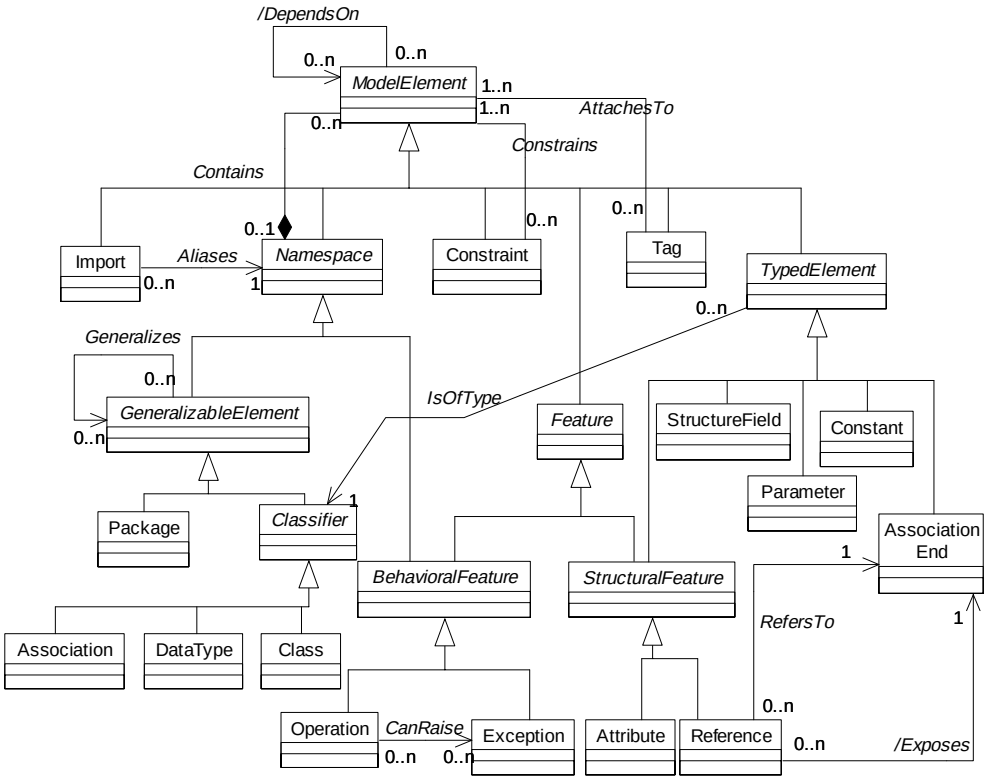


Figure 6. Le MOF

Un des avantages de cet alignement est que nombre de mécanismes initialement développés pour UML ont pu être aisément intégrés dans la spécification du MOF. On peut citer par exemple OCL [104] (Object Constraint Language), langage de contraintes permettant de définir des assertions sur des modèles UML, réutilisé au niveau du MOF. De manière pratique, un AGL UML va pouvoir être utilisé pour manipuler des méta-modèles basés sur le MOF. Cela implique une utilisation particulière de l'outil puisqu'on doit alors se restreindre à l'emploi des concepts communs aux deux formalismes (pas de cas d'utilisation, de diagramme d'états ou de classe-associations). L'AGL Rose, édité par la société Rational, dispose ainsi d'un module complémentaire développé par la société Unisys pour l'import/export de modèles UML et de méta-modèles MOF au format XMI.

Typiquement, un méta-modèle basé sur le MOF est un paquetage (instance de *Package*) qui contient un ensemble de classes (instances de *Class*), reliées entre elles par des associations (instances d'*Association*) et potentiellement regroupées en sous-paquetages (instances de *Package*). Sur une classe on peut définir des attributs (instances d'*Attribute*) ainsi que des opérations (instances d'*Operation*). Les classes sont reliées aux associations par des références (instances de *Reference*) vers des extrémités d'association (instance d'*AssociationEnd*). Enfin, on peut préciser le méta-modèle en ajoutant des contraintes (instances de *Constraint*) sur chacun des éléments. Dans l'exemple suivant (Figure 7), on a défini un méta-modèle MOF extrêmement simplifié destiné à la représentation de bases de données.

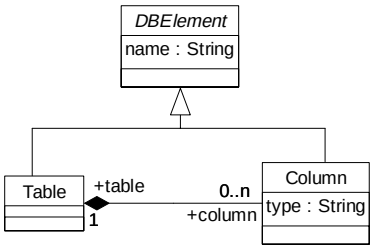


Figure 7. Méta-modèle MOF d'une base de donnée

Un certain nombre d'outils permettent de définir et de manipuler des méta-modèles basés sur le MOF. On peut citer par exemple :

- M3J : un outil développé par Xavier Blanc, au cours d'un partenariat entre le Lip6 et EDF R&D, pour la définition graphique de méta-modèles et la génération des interfaces IDL et de la DTD XMI (http://www-src.lip6.fr/meta/Projets/M3J/m3j.html).
- Universalis : un référentiel de modèle développé par France Telecom R&D basé sur le MOF et pouvant donc abriter tout modèle dont le méta-modèle est basé sur le MOF (http://universalis.elibel.tm.fr/index.html).
- dMOF : un outil développé au sein du DSTC (Distributed System Technology Centre) en Australie pour la définition et la gestion de méta-modèles basés sur le MOF (http://www.dstc.edu.au/Products/CORBA/MOF/).

2.2.2. Un formalisme propriétaire : les sNets

Les sNets sont le résultat d'une collaboration entre la société Sodifrance et l'Université de Nantes. Ces travaux ont été finalisés par Richard Lemesle dans le cadre de son stage de DEA et de sa thèse [49]. Les sNets définissent un formalisme minimal. Au coeur du formalisme sNets on trouve la notion de réseau sémantique, c'est-à-dire un ensemble de noeuds et de liens. Ces deux concepts (noeud et lien) constituent la partie réflexive du méta-méta-modèle présenté Figure 8.

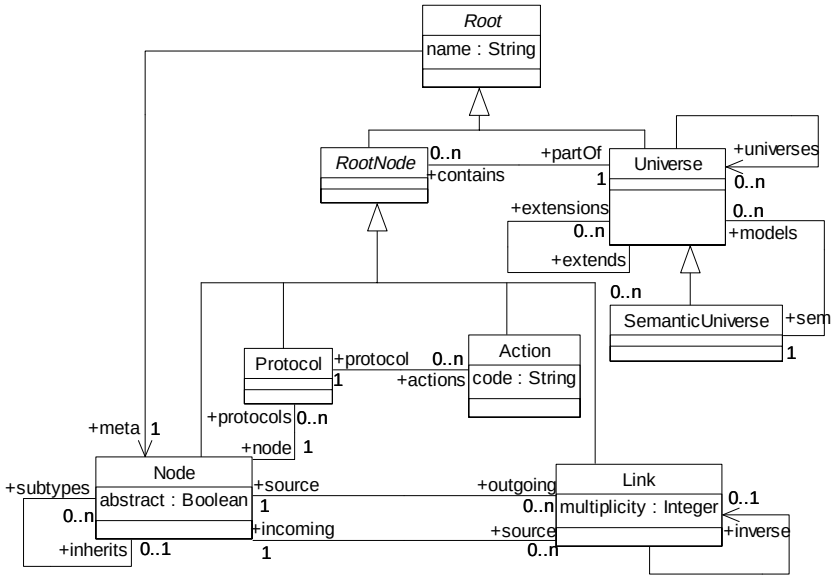


Figure 8. Le méta-méta-modèle complet des sNets

Dans l'environnement sNets un méta-modèle est un univers sémantique (instance de *Semantic Universe*). Il contient un ensemble de nœuds (instances de *Node*) reliés entre eux par des liens (instances de *Link*). Ces éléments peuvent être regroupés à l'intérieur de sous-univers (instances de *Semantic Universe*). Sur les nœuds, on peut définir des actions (instance d'*Action*), celles-ci étant regroupées en protocoles (instance de *Protocol*). Dans l'exemple suivant, on a défini un méta-modèle sNets extrêmement simplifié de base de donnée (Figure 9).

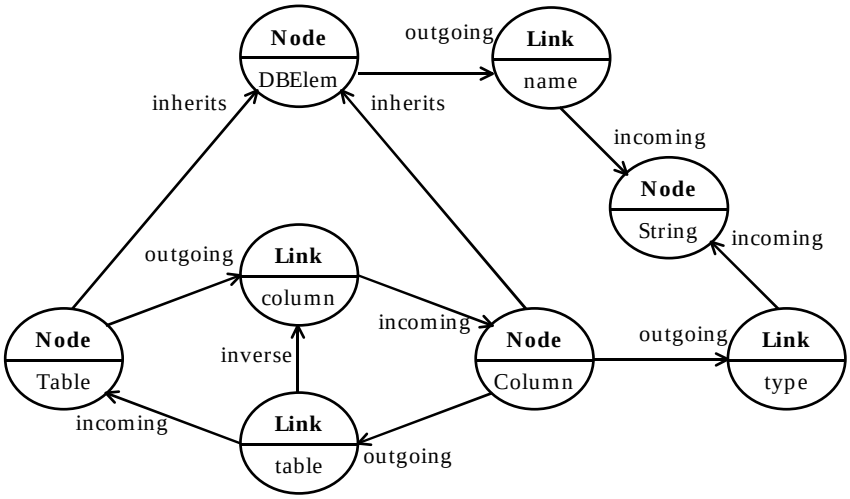


Figure 9. Méta-modèle sNets d'une base de donnée

Les sNets définissent un certain nombre de mécanismes similaires à ceux définis dans le MOF (héritage, extension). Par contre, ils définissent également les relations entre les niveaux de modélisation. Ainsi la relation entre un modèle (instance de *Universe*) et son méta-modèle (instance de *Semantic Universe*) est-elle explicitée par le lien *sem*. De la même façon la relation entre un nœud et son méta-nœud est-elle définie par le lien *meta*. Cela permet par exemple de spécifier que le méta-nœud d'un nœud doit appartenir à l'univers sémantique (ou à l'un des univers sémantiques que ce dernier étend) sur lequel est basé l'univers contenant le nœud. Cette définition explicite du concept d'instanciation permet de rationaliser les relations entre niveaux de modélisation, ainsi que d'éliminer certaines confusions quant à l'usage abusif du terme instanciation ([6], [7]).

Il existe deux implémentations des sNets, l'une en Smalltalk et l'autre en Java. Ces implémentations intègrent un moteur d'interprétation, la sMachine [5], ainsi qu'une interface de navigation dans les modèles, le sBrowser. Le choix du langage a une importance. Les sNets ne fournissant pas de langage d'action spécifique, le corps des actions est écrit en utilisant le langage avec lequel est implémenté la sMachine, c'est-à-dire Smalltalk ou Java. La société Sodifrance a utilisé ces implémentations pour développer un certain nombre de produits :

- Semantor : un outil de rétro-analyse de code Cobol
- Scriptor-Generation : un outil de génération de code
- Scriptor-Transformation : un outil de transformation de modèles

2.3. Des formats d'échanges normalisés

2.3.1. XMI

XMI est une spécification de l'OMG [75]. L'objectif de XMI est de permettre de sérialiser et d'échanger des méta-modèles MOF et des modèles basés sur ces méta-modèles

sous la forme de fichier en utilisant des dialectes XML. XMI ne définit pas un dialecte XML unique mais un ensemble de règles permettant la production d'une DTD à partir d'un méta-modèle basé sur le MOF. Ainsi, une DTD a pu être produite pour le MOF lui-même, puisqu'il s'auto-définit. Le méta-modèle UML peut aussi bien être échangé sous la forme d'un fichier XMI conforme à la DTD du MOF, que d'une DTD. Un modèle UML peut être communiqué comme un fichier XML conforme à la DTD XMI produite à partir d'UML (Figure 10).

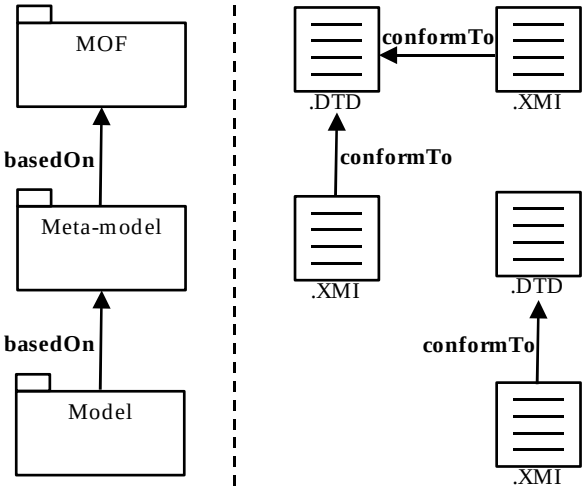


Figure 10. Architecture MOF / architecture XMI

La DTD XMI du méta-modèle de base de données basé sur le MOF que nous avons présenté précédemment (Figure 7) sera alors la suivante (Figure 11). Pour des raisons de place nous n'affichons qu'une petite partie de la DTD complète.

```
<!ELEMENT XMI (XMI.header, XMI.content?, XMI.difference*, XMI.extensions*) >
<!ATTLIST XMI
            xmi.version CDATA #FIXED "1.0"
            timestamp CDATA #IMPLIED
            verified (true | false) #IMPLIED
>
…
<!— Divers paramétrages + definition des types de base -->
…
<!ELEMENT DBElem.name (#PCDATA | XMI.reference)* >
<!ELEMENT DBElem (DBElem.name?, XMI.extension*)? >
<!ATTLIST DBElem
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT Table.column (Column)* >
<!ELEMENT Table (DBElem.name?, XMI.extension*, Table.columnl*)? >
<!ATTLIST Table
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT Column.type (#PCDATA | XMI.reference)* >
<!ELEMENT Column.table (Table)? >
<!ELEMENT Column (DBElem.name?, Column.type?, XMI.extension*, Column.table?)? >
<!ATTLIST Column
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT DBMetaModel ((Table | Column )*) >
<!ATTLIST DBMetaModel
            %XMI.element.att;
            %XMI.link.att;
>
```

Figure 11. Exemple de DTD produite à partir d'un méta-modèle MOF

Des travaux sont actuellement en cours à l'OMG afin de déterminer des règles de production de schémas XML à partir de méta-modèles basés sur le MOF [74]. Une proposition est actuellement en cours de finalisation, l'échéance fixée pour la recommandation étant juillet 2002. Il s'agit de remplacer les DTDs par les schémas XML. Ces derniers présentent de nombreux avantages. Tout d'abord un schéma XML est écrit dans un langage XML (ce qui n'est pas le cas le cas d'une DTD). De plus un schéma XML fournit davantage de mécanismes de structuration de l'information (typage, multiplicité, héritage, etc.) qu'une DTD.

2.3.2. XML-sNets

XML-sNets est le format d'échange pour l'ensemble des modèles de l'architecture sNets. Contrairement à XMI il ne définit qu'une et une seule DTD (Figure 12).

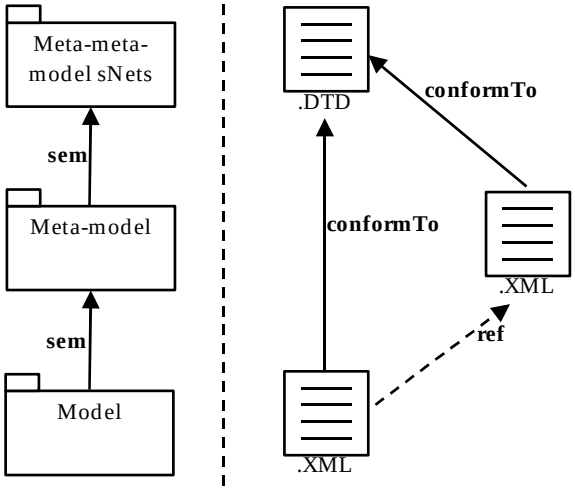


Figure 12. Architecture sNets / architecture XML-sNets

Cette DTD ne distingue que quatre types d'éléments : des univers, des noeuds, des attributs et des liens (Figure 13).

```
<!ELEMENT UNIVERSE (ATTRIBUTE | LINK | NODE | UNIVERSE)*>
<!ATTLIST   UNIVERSE
            NAME            CDATA #REQUIRED
            ID              CDATA #REQUIRED
            SEMANTIC        CDATA #REQUIRED
            EXTENDS         CDATA #IMPLIED
            TYPE            CDATA #REQUIRED>

<!ELEMENT NODE (ATTRIBUTE | LINK)*>
<!ATTLIST   NODE
            NAME            CDATA #REQUIRED
            ID              CDATA #REQUIRED
            TYPE            CDATA #REQUIRED>

<!ELEMENT ATTRIBUTE EMPTY>
<!ATTLIST   ATTRIBUTE
            NAME            CDATA #REQUIRED
            TYPE            CDATA #REQUIRED
            VALUE           CDATA #REQUIRED>

<!ELEMENT LINK EMPTY>
<!ATTLIST   LINK
            NAME            CDATA #REQUIRED
            DESTINATION CDATA #REQUIRED>
```

Figure 13. L'unique DTD XML-sNets

Le code XML permettant d'échanger le méta-modèle de base de données sNets que nous avons présenté précédemment (Figure 9) sera alors le suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE UNIVERSE SYSTEM "sem.dtd">
<UNIVERSE NAME="DBMetaModel" ID="#47017262" SEMANTIC="world.semantic"
                EXTENDS="world.commonSem" TYPE="world.commonSem.SemanticUniverse">
        <NODE NAME="DBElem" ID="#47016010" TYPE="world.semantic.Node">
                <LINK NAME="outgoing" DESTINATION="#47006517"/>
                <LINK NAME="subtypes" DESTINATION="#47028242"/>
                <LINK NAME="subtypes" DESTINATION="#47025457"/>
                <LINK NAME="subtypes" DESTINATION="#47031988"/>
        </NODE>
        <NODE NAME="Table" ID="#47025457" TYPE="world.semantic.Node">
                <LINK NAME="outgoing" DESTINATION="#47000038"/>
                <LINK NAME="outgoing" DESTINATION="#47011059"/>
                <LINK NAME="outgoing" DESTINATION="#47026749"/>
                <LINK NAME="inherits" DESTINATION="#47016010"/>
                <LINK NAME="source" DESTINATION="#47000828"/>
                <LINK NAME="source" DESTINATION="#47022700"/>
        </NODE>
        <NODE NAME="Column" ID="#47031988" TYPE="world.semantic.Node">
                <LINK NAME="outgoing" DESTINATION="#47032617"/>
                <LINK NAME="outgoing" DESTINATION="#47022700"/>
                <LINK NAME="inherits" DESTINATION="#47016010"/>
                <LINK NAME="source" DESTINATION="#47029190"/>
                <LINK NAME="source" DESTINATION="#47011059"/>
        </NODE>
        <NODE NAME="name" ID="#47006517" TYPE="world.semantic.Link">
                <LINK NAME="source" DESTINATION="#47016010"/>
                <LINK NAME="incoming" DESTINATION="world.commonSem.String"/>
                <ATTRIBUTE NAME="multiplicity" TYPE="world.commonSem.Integer" VALUE="1"/>
        </NODE>
        <NODE NAME="type" ID="#47032617" TYPE="world.semantic.Link">
                <LINK NAME="source" DESTINATION="#47031988"/>
                <LINK NAME="incoming" DESTINATION="world.commonSem.String"/>
                <ATTRIBUTE NAME="multiplicity" TYPE="world.commonSem.Integer" VALUE="1"/>
        </NODE>
        <NODE NAME="column" ID="#47011059" TYPE="world.semantic.Link">
                <LINK NAME="source" DESTINATION="#47025457"/>
                <LINK NAME="incoming" DESTINATION="#47031988"/>
                <ATTRIBUTE NAME="multiplicity" TYPE="world.commonSem.Integer" VALUE="-1"/>
                <LINK NAME="inverse" DESTINATION="#47022700"/>
        </NODE>
        <NODE NAME="table" ID="#47022700" TYPE="world.semantic.Link">
                <LINK NAME="source" DESTINATION="#47031988"/>
                <LINK NAME="incoming" DESTINATION="#47025457"/>
                <ATTRIBUTE NAME="multiplicity" TYPE="world.commonSem.Integer" VALUE="1"/>
                <LINK NAME="inverse" DESTINATION="#47011059"/>
        </NODE>
</UNIVERSE>
```

Figure 14. Exemple de fichier XML produit à partir d'un méta-modèle sNets

2.4. Des techniques et des outils de transformation

2.4.1. XSLT

XSLT [103] (XSL Transformations) est une recommandation du W3C. C'est un langage basé sur XML dédié à la transformation de fichiers XML. Il généralise le langage XSL [100] (eXtensible Stylesheet Language), autre recommandation du W3C, destiné à la mise en page pour affichage au format HTML de documents XML. XSLT était initialement limité à la transformation de XML vers XML, mais aujourd'hui les formats cibles ont été étendus à tout format textuel.

Quand on met en oeuvre XSLT, on manipule donc trois fichiers différents :

- le fichier source, respectant un langage XML quelconque,
- le fichier cible, qui est un fichier au format texte,
- le fichier contenant les règles de transformation écrit en XSLT.

L'application des règles de transformation sur le fichier source afin de produire le fichier cible se fait par l'intermédiaire d'un processeur XSLT (Figure 15).

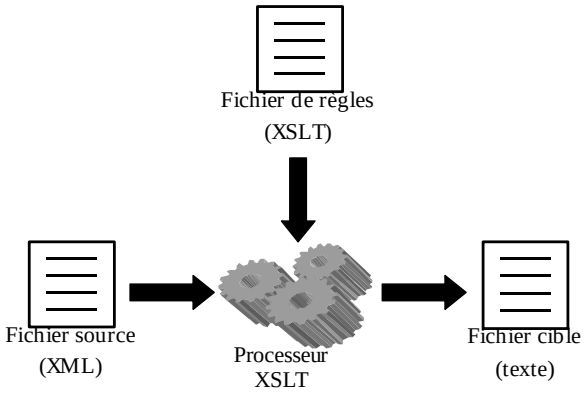


Figure 15. Mise en oeuvre de XSLT

Un fichier de règles XSLT définit donc une procédure de génération de texte à partir des informations contenues par le fichier source. Cette procédure est spécifiée à l'aide de mécanismes de contrôles. Un certain nombre de structures ont ainsi été définies pour décrire des boucles ou des expressions conditionnelles. On a par exemple les balises suivantes : *template*, *choose*, *if*, *when*, *for-each*. Ces structures de contrôles sont évaluées à partir des données contenues dans les champs du fichier source. Les requêtes permettant d'accéder à ces champs sont spécifiées à l'aide d'expressions XPath [102], permettant d'indiquer la navigation dans l'arborescence du fichier source. Une expression XPath définit un parcours dans le fichier XML à partir du nœud courant (le contexte). Chaque étape de la procédure de génération peut produire une portion de texte. Celui-ci peut aussi bien définir des parties textuelles que des accès à des informations contenues dans le fichier source. Ces dernières sont également déterminées en utilisant des expressions XPath. L'exemple suivant (Figure 16) montre un exemple de fichier XSLT transformant un fichier XML stockant des classes en un fichier XML stockant des tables.

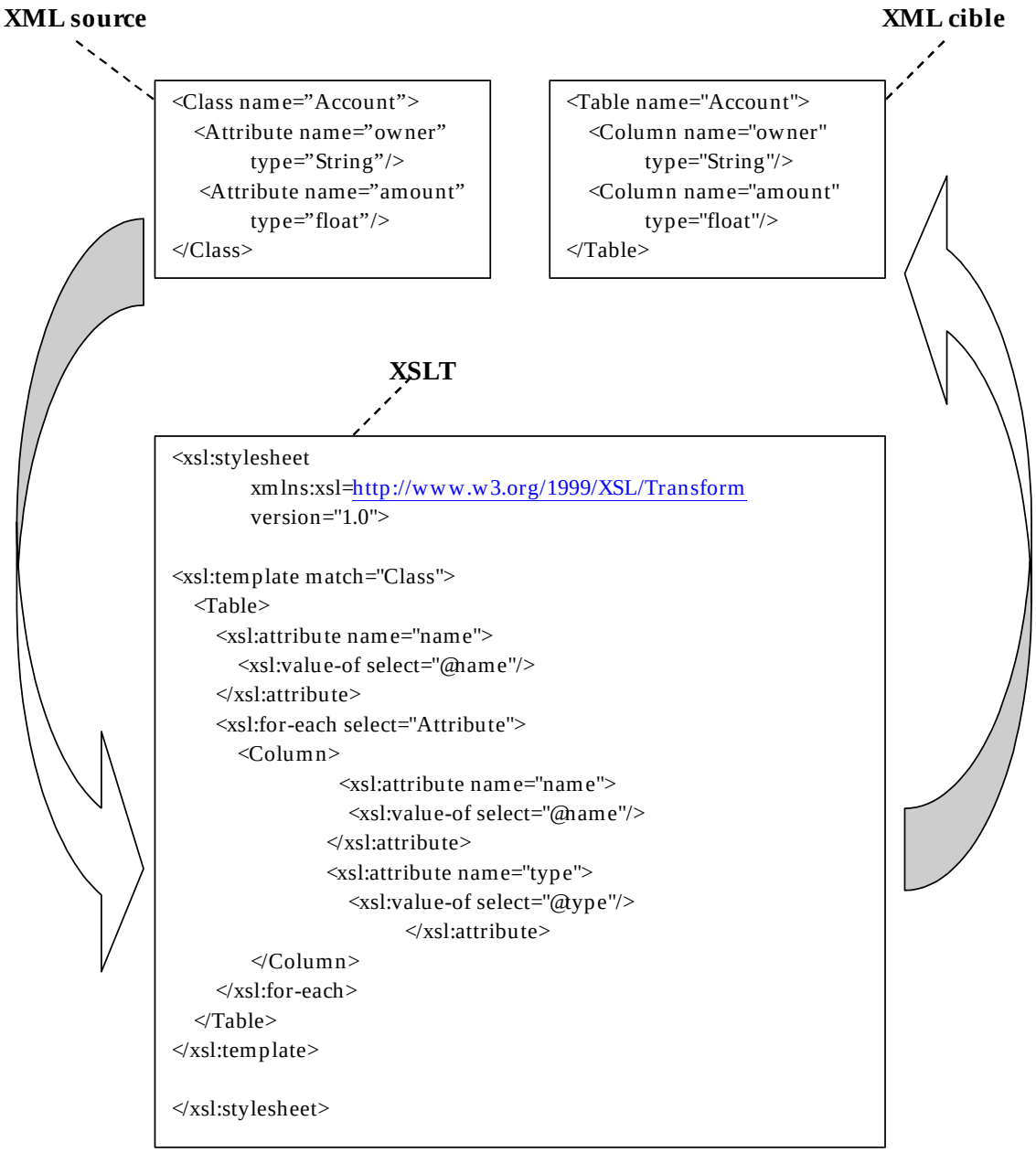


Figure 16. Exemple de fichier XSLT

Un des principaux avantages de XSLT est qu'il se base sur XML. On peut ainsi le manipuler comme on le ferait avec n'importe quel fichier XML. On bénéficie de l'ensemble des outils (éditeurs, parseurs, etc.) et des techniques développés autour de XML. En particulier on peut appliquer une transformation XSLT à un fichier XSLT.

Par contre, il présente l'inconvénient majeur de s'appuyer sur la syntaxe concrète des fichiers XML source et cible. Or, on peut associer différents formats XML (DTD ou schéma) à un même méta-modèle. Cela dépend des stratégies que l'on décide d'appliquer. Par exemple, lors de la tentative de définition d'un dialecte XML pour l'échange de schémas et de données EXPRESS, deux stratégies concurrentes (« early-binding » et « late-binding ») ont été étudiées [41]. A l'OMG le problème semble avoir été résolu par l'adoption de XMI,

qui impose un ensemble de stratégies. Toutefois, XMI évolue. On en est aujourd'hui à la version 1.3. Ces évolutions ne sont pas forcément synchronisées avec celles du MOF. Un même modèle peut donc être sérialisé selon différentes versions de la norme XMI, et donc dans des formats XML différents. Les transformations XSLT spécifiées pour une version de XMI ne sont donc pas forcément valides pour une autre. A chacune des versions peut alors correspondre un fichier de règles particulier, alors que les concepts manipulés sont identiques. Pour une transformation entre deux mêmes méta-modèles, on risque donc de se retrouver confronté à un nombre important de fichiers XSLT, spécifiant des correspondances identiques entre les deux méta-modèles, mais adaptés à des versions particulières de XMI.

2.4.2. Scriptor-Generation

Scriptor-Generation (Scriptor-G) est un outil de génération de code développé et distribué par la société Sodifrance [89]. Il est implémenté en Java et il se base sur les sNets, ce qui lui donne une grande souplesse d'adaptation. Le noyau de Scriptor-G est constitué d'un méta-méta-modèle, et non d'un méta-modèle comme la plupart des outils de génération. Le moteur de génération est donc générique. Toutefois, chaque version de Scriptor-G correspond à un et un seul méta-modèle. Ce dernier est un composant interchangeable. Ainsi il y a des versions de Scriptor-G dédiées à la génération de code à partir de modèles UML, et d'autres versions dédiées à des formalismes moins répandus ou propriétaires (voir la partie Réalisations industrielles).

Le principe de Scriptor-G est le suivant (Figure 17). On définit des actions sémantiques sur les entités d'un méta-modèle. Ces actions seront ensuite appliquées sur un modèle afin de produire un fichier texte résultat. Elles sont spécifiées en Java, ce qui permet de bénéficier de toutes les opérations élémentaire offertes par ce langage. La navigation dans le modèle source se fait par un ensemble d'APIs Java définies sur le méta-modèle. On peut comparer ce mécanisme à une première implémentation de la JSR 40, initiative de Sun pour la spécification d'une interface Java de gestion des méta-données [93].

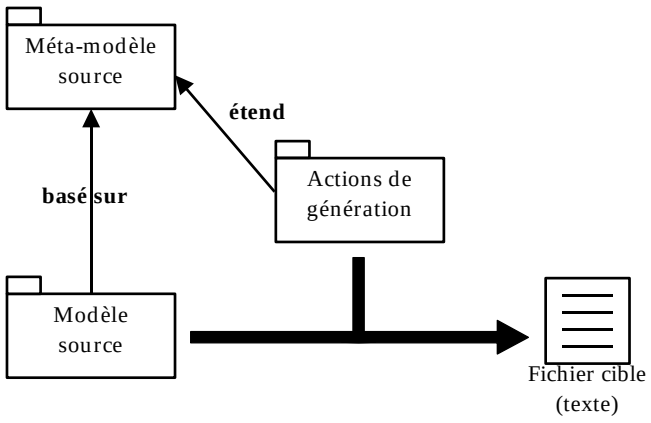


Figure 17. Principes de Scriptor-G

Scriptor-G se décompose en quatre composants majeurs. Au cœur de Scriptor-G, il y a le méta-méta-modèle, qui permet de manipuler le méta-modèle, les modèles et les règles. Il y a ensuite l'éditeur de règles qui permet de saisir les règles de génération. Le moteur de

génération se charge d'appliquer ces règles. Enfin les lecteurs permettent d'importer des modèles à partir d'un référentiel quelconque (fichier, base de données, application, etc.). Ainsi, on peut utiliser différents formats d'import sans que les règles de génération soient affectées (Figure 18).

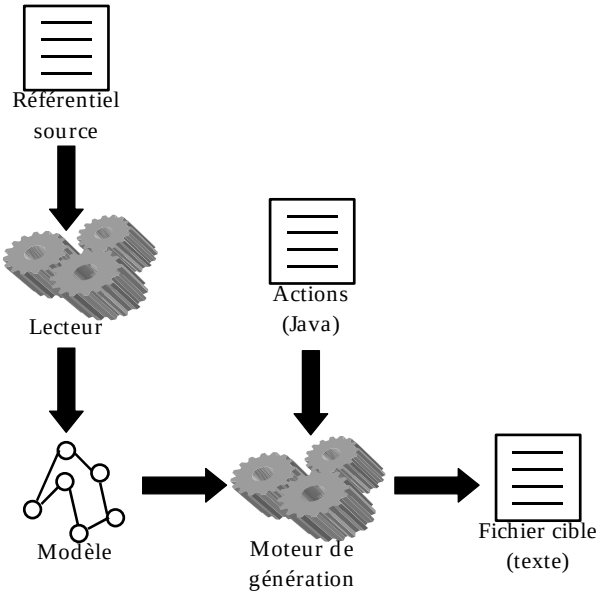


Figure 18. Mise en oeuvre de Scriptor-G

Les actions sémantiques définies au sein de Scriptor-G peuvent être des scripts, des macros ou des services. Les scripts contiennent du texte qui sera généré tel quel, ainsi que des appels à d'autres scripts ou à des macros. Cette invocation se fait en insérant le nom de l'opération à activer à l'intérieur du script et en l'encadrant par des double crochets (*[[nomOperation]]*). Les macros retournent des chaînes de caractère et contiennent du code Java. Elles permettent de naviguer dans le modèle et de disposer de la richesse du langage Java pour la manipulation de chaînes de caractères. Enfin, les services permettent de retourner des collections d'objets. Des mécanismes ont été développés pour manipuler les résultats de ces services. Ainsi, dans l'exemple ci-dessous (Figure 19), on applique un script à l'ensemble des objets d'une collection calculée par le service *attributes* grâce à la méthode *apply*.

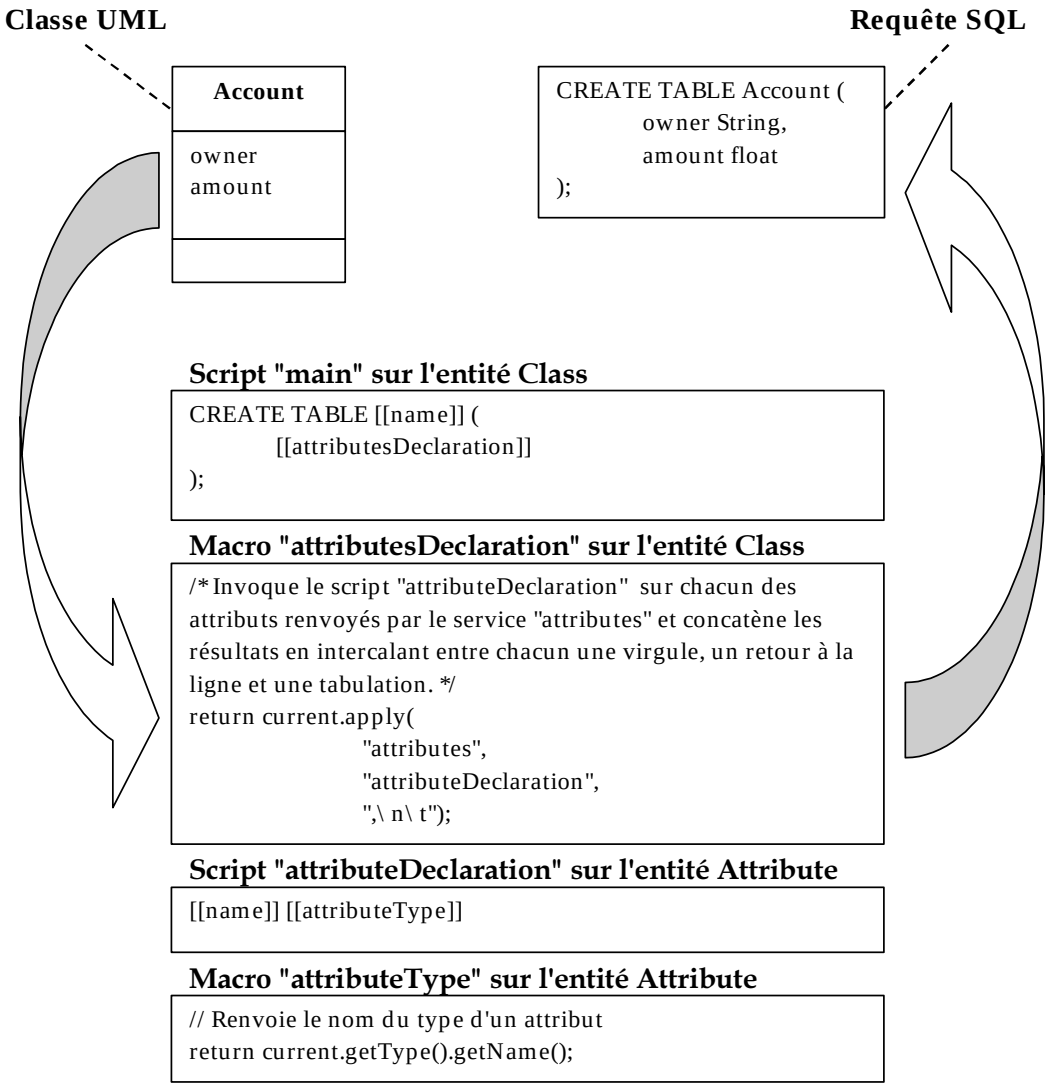


Figure 19. Génération d'une requête SQL à partir d'une classe UML

Un des principaux avantages de Scriptor-G tient au fait que la navigation dans le modèle source ne dépend en aucune façon de son format d'échange. Par contre, le texte produit reste dépendant d'un format particulier puisque la génération se base directement sur la syntaxe concrète du formalisme cible.

2.4.3. Scriptor-Transformation

Scriptor-Transformation (Scriptor-T) est développé et distribué par la société Sodifrance [90]. Il est directement issu des travaux de Richard Lemesle sur la transformation de modèles, un premier prototype ayant été développé durant sa thèse [48]. Cet outil se base sur les sNets. Il permet de spécifier des règles de transformation entre un méta-modèle source et un méta-modèle cible, et de les appliquer sur un modèle source afin de produire un modèle cible (Figure 20).

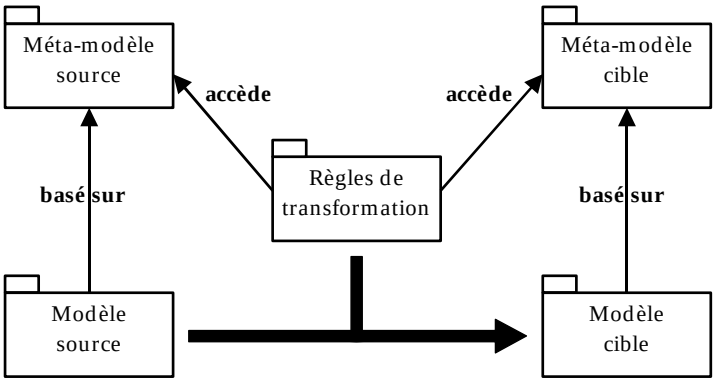


Figure 20. Principes de Scriptor-T

Tout comme Scriptor-G, Scriptor-T est divisé en un certain nombre de composants. Au cœur de l'outil on trouve le méta-méta-modèle sNets qui permet de manipuler les modèles et les méta-modèles. Il y a le moteur de transformation qui est un moteur d'inférences. Enfin il y a des composants d'import (lecteurs) et d'export (écrivains) qui permettent de lire et de sauvegarder des modèles dans divers formats (Figure 21).

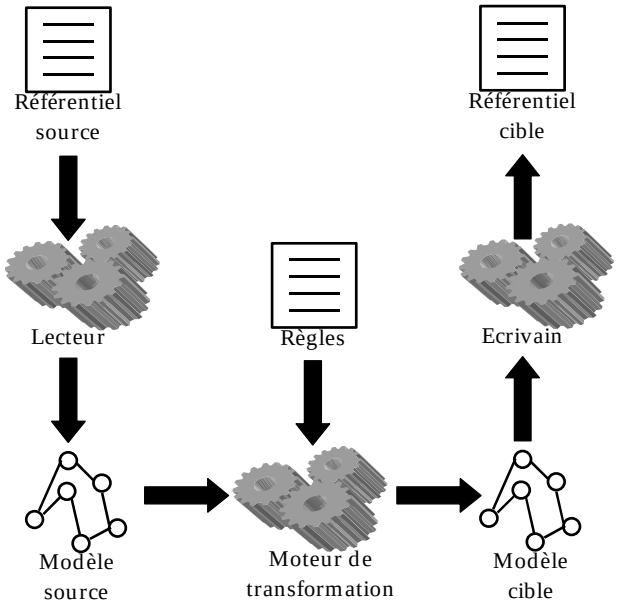


Figure 21. Mise en oeuvre de Scriptor-T

Les règles sont écrites dans un langage de type condition/conclusion (voir Figure 22). Une condition peut aussi bien contenir des assertions sur le modèle source que sur le modèle cible. La conclusion permet de créer et de lier des éléments dans le modèle cible. De plus, il est possible de créer des relations temporaires de coréférence entre des éléments du modèle source et du modèle cible. Ces relations peuvent également être réutilisées dans les conditions.

L'avantage de Scriptor-T est que les règles de transformation ne sont en aucun cas dépendantes d'exigences particulières au support de persistance, tant au niveau de la source que de la cible.

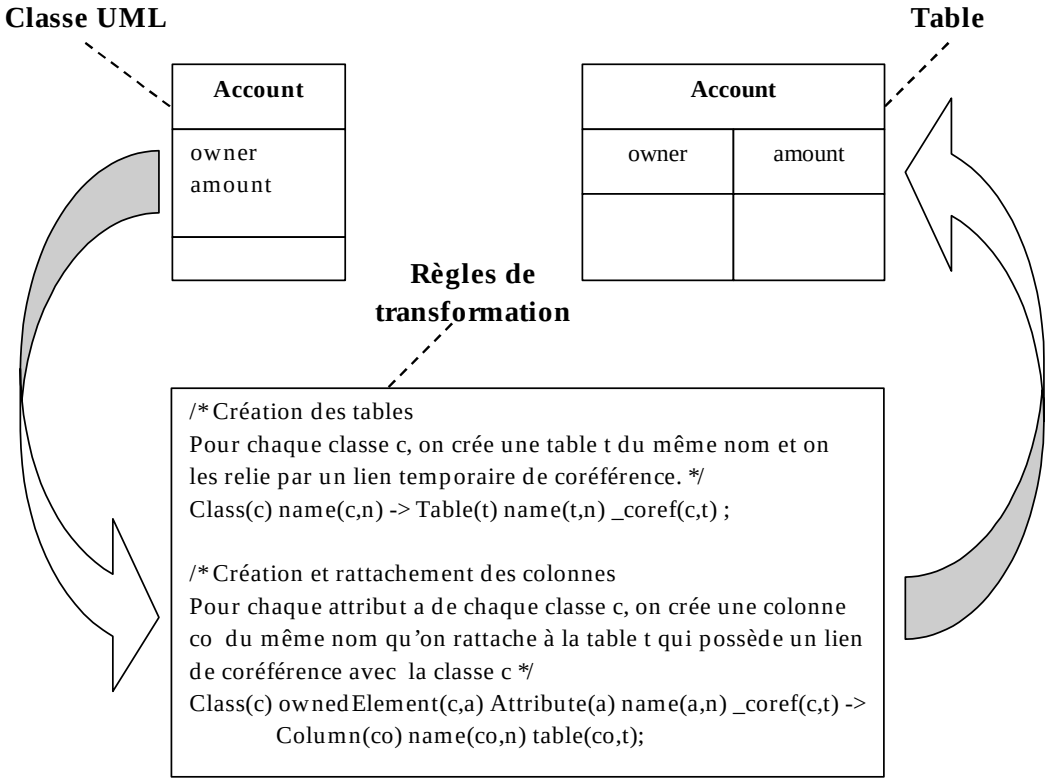


Figure 22. Règles de transformation d'une classe UML en Table

2.5. L'émergence du MDA

Le MDA est issu du constat que chaque nouvelle évolution technologique entraîne un coût significatif. Ce dernier découle de la migration de tout ou partie du système, et de la définition de mécanismes d'interopérabilité entre les composants basés sur cette nouvelle technologie et l'existant (« legacy systems »). Une partie importante de ce coût est consacrée à la réécriture des règles et des structures de données métier. Cette réécriture peut être assistée par des mécanismes complexes de rétro-documentation et de transformation, mais la majeure partie du travail demeure manuelle. Cette situation devient d'autant moins supportable pour les industriels que le rythme d'évolution des technologies s'est singulièrement accéléré ces dernières années, avec l'apparition de plates-formes telles que J2EE ou .NET.

Le MDA est la réponse de l'OMG à cet état de fait. Son objectif est de permettre l'intégration et l'interopérabilité, non seulement avec l'ensemble des systèmes existants, mais également avec ceux qui existeront. Pour cela, l'OMG ne place plus l'interopérabilité au niveau du code, entre composants logiciels exécutables, comme c'était le cas avec CORBA. L'interopérabilité est définie au niveau des modèles. De plus, les modèles métier (PIM) sont séparés des modèles d'implémentation (PSM) qui prennent en compte les spécificités dues à la plate-forme d'exécution. Les deux avancées importantes du MDA sont donc d'une part

l'utilisation systématique des techniques de modélisation et de méta-modélisation, et d'autre part la séparation des aspects relatifs aux domaines de ceux relatifs à la plate-forme.

Desmond D'Souza [22] a identifié trois dimensions dans l'architecture des modèles représentant une entreprise

- La dimension horizontale : au sein d'une même entreprise cohabitent différentes vues partielles du système, chaque vue correspondant à un domaine différent ayant ses objectifs propres et sa terminologie particulière. Chaque vue se traduit par des modèles particuliers, et éventuellement des méta-modèles spécifiques.
- La dimension verticale : à l'intérieur d'un même domaine, on peut trouver différents niveaux d'abstraction.
- La dimension des variantes : au sein de cette architecture peuvent cohabiter différentes versions de modèles, les modèles représentant le système tel qu'il a été, tel qu'il est, tel qu'il sera ou tel qu'il pourrait être.

La première problématique du MDA concerne l'organisation des modèles, et donc des méta-modèles. En effet les différents domaines ne sont pas totalement disjoints. Il existe de nombreuses relations entre eux. Il est donc nécessaire d'expliciter ces relations, et ce au plus haut niveau d'abstraction afin de faciliter l'interopérabilité. Une seconde problématique est celle du passage d'un niveau d'abstraction à l'autre, par exemple d'un PIM à un PSM. Le MDA prône une automatisation, au moins partielle, du maintien de la cohérence entre modèles de niveaux d'abstraction différents. Comme des méta-modèles différents peuvent être utilisés à chaque niveau d'abstraction, les techniques de transformation de modèles sont appelées à jouer un rôle majeur au sein du MDA. Le raffinement d'une spécification ne se fait plus par la composition de classes, mais par des transformations successives de modèles [3].

2.6. Conclusion

Dans cette section nous avons présenté un panorama des différentes composantes de la méta-modélisation. Plutôt que de dresser un état de l'art exhaustif, nous avons limité notre étude aux différentes techniques et applications que nous avons pu mettre en œuvre. Nous avons ainsi introduit les mécanismes de représentation, de communication et de manipulation de l'information sur lesquels se basent l'ensemble de nos travaux et réalisations.

Un méta-modèle définit un langage de représentation pour un domaine d'intérêt particulier. Ce langage est spécifié sous la forme d'un ensemble de concepts (*Class* en MOF, *Node* en sNets), de relations (*Association* en MOF, *Link* en sNets) et d'un certain nombre d'autres éléments (tels que des assertions OCL en MOF). Il existe donc de nombreux points communs entre les méta-modèles et les langages de programmation basés sur des grammaires formelles, ainsi qu'entre les méta-modèles et les ontologies. Dans cette

conclusion, nous tentons de fournir quelques éléments de comparaison entre ces formalismes.

Le terme « langage » est utilisé jusque dans l'acronyme du plus célèbre méta-modèle : UML (Unified Modeling Language). L'architecture de modélisation peut être comparée à celle des langages formels. La grammaire d'un langage, par exemple Pascal, est défini à partir de la notation EBNF (Extended Backus Naur Form). Ce langage va servir à définir des programmes, chaque programme pouvant donner lieu à une infinité d'exécutions.

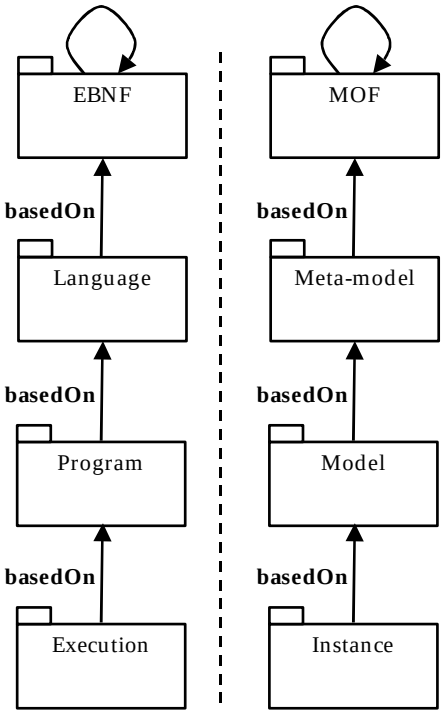


Figure 23. Architecture des langages / architecture des modèles

La notation EBNF permet de spécifier la syntaxe concrète d'un langage, c'est-à-dire l'ensemble des suites de caractères valides. En cela elle est différente du MOF, qui s'attache plus à la définition des concepts et de leurs relations. La notation EBNF permet d'associer des séquences de caractères à des méta-identifieurs. Par contre elle ne permet pas d'expliciter des relations, de les organiser en graphes d'héritage ou de leur affecter des propriétés. De nombreux travaux proposent la représentation des expressions syntaxiques sous la forme d'arbres. Par contre un modèle est le plus souvent considéré comme un graphe.

Pour manipuler un langage de programmation (par exemple pour spécifier la sémantique d'exécution), on se base sur sa syntaxe abstraite. La syntaxe abstraite d'un langage définit l'ensemble des structures du langage, à la différence de la syntaxe concrète qui décrit les séquences de caractères valides. La syntaxe abstraite peut être exprimée en EBNF, en omettant le plus souvent les symboles terminaux. Dans leur papier sur les langages de modélisation [33], David Harel et Bernhard Rumpe assimilent les concepts de syntaxe abstraite et de méta-modèle. Ainsi, de la même façon qu'on peut avoir plusieurs

syntaxes concrètes pour une même syntaxe abstraite, on pourrait dire que XMI et la représentation graphique sont deux syntaxes concrètes du méta-modèle UML.

On voit donc que les points communs entre méta-modèles et langages sont nombreux. Les principales différences proviennent du formalisme de définition. La notation EBNF permet de préciser la décomposition d'une entité en un certain nombre d'entités de moins haut niveau. Le MOF définit également cette relation de composition, mais il introduit également un certain nombre d'associations supplémentaires. On a donc d'un côté une architecture arborescente, et de l'autre une organisation en graphe.

Une des définitions les plus utilisée pour le terme ontologie est celle de Thomas Gruber : « une ontologie est une spécification explicite d'une conceptualisation » [31]. Une conceptualisation est une vue abstraite, simplifiée du monde que l'on souhaite représenter. Cette vue du monde est décrite comme un ensemble d'objets (qui forment l'univers du discours) accompagnés de leurs inter-relations. Michael Uschold et Michael Gruninger [98] ont identifié un certain nombre d'usages pour les ontologies. Les ontologies peuvent permettre de faciliter la communication entre les personnes, d'assurer l'interopérabilité entre composants logiciels, et d'assister l'ingénierie des systèmes. De leur côté, Balakrishnan Chandrasekaran et al. [13] distinguent deux apports importants des ontologies : la clarification des structures de la connaissance par l'analyse ontologique et le partage de cette connaissance. Dans les deux cas, l'accent est mis sur la communication et le partage. Pour être utilisable, une conceptualisation doit être partagée. Une ontologie est donc le résultat d'un consensus autour d'un point de vue particulier.

En fait, on s'aperçoit qu'il n'y a pas de divergences flagrantes, tant dans les objectifs que dans les techniques, entre ontologies et méta-modèles. On peut positionner les méta-modèles comme une forme particulière d'ontologie, qui intègrerait la notion d'un méta-méta-modèle unique et d'une architecture en couches. Ce rapprochement entre ontologies et méta-modèles a déjà été étudié dans [2] où les deux termes sont utilisés de façon indifférenciée.

3. Différentes utilisations du concept de processus

3.1. Introduction

Le concept de processus est utilisé dans des domaines très variés. Dans nos travaux, tant industriels qu'universitaires, nous avons eu l'occasion d'échanger sur ce sujet avec des personnes issues de quelques-uns de ces domaines. Dans chacun d'entre eux, la notion de processus est un élément important mais est utilisée avec des objectifs particuliers. Toutefois, nous avons toujours réussi à dialoguer, car nos définitions respectives du terme processus n'étaient pas très éloignées, même s'il existe quelques différences, ainsi que le montrent les exemples suivants :

- *A coordinated set of activities that are connected in order to achieve a common goal* (WfMC, [107])
- *All the « real-world » elements involved in the development and maintenance of a product, i.e. artifacts, production support, activities, agents and process support* (Jean-Claude Derniame et al., **[20]**)
- *A series of activities that are linked to perform a specific objective* (CAM-I, Consortium for Advanced Manufacturing International)
- *Set of interrelated or interacting activities which transforms inputs into outputs* (ISO, [37])
- *A process is a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs* (BPMI.org, [9])

Avant d'entamer une étude plus complète de différents formalismes dédiés à la représentation de processus, il nous a semblé important de présenter l'emploi de ce concept dans quelques domaines. Faire une liste exhaustive des domaines utilisant cette notion dépasserait le cadre de cette thèse. Nous nous sommes volontairement restreints à ceux dans lesquels nos travaux nous ont amenés à faire une incursion. Il s'agit de :

- L'ingénierie système,
- Le contrôle de gestion et l'organisation,
- La gestion de la qualité,
- Le développement logiciel.

Nous ne prétendons aucunement à une expertise quelconque dans ces différents domaines. Nous tentons, à partir des éléments qui nous apparaissent pertinents, de dresser un panorama de diverses utilisations du concept de processus dans des corporations et des contextes différents.

3.2. Dans le domaine de l'ingénierie système

Selon la définition donnée par l'ISO, un système est un objet composé d'éléments interagissant [40]. Ces éléments peuvent être du matériel, du logiciel, des ressources humaines, des processus et des sous-systèmes. Les systèmes sont créés et utilisés pour fournir des produits et des services dans des environnements définis. Pour illustrer la notion de système, l'ISO cite l'exemple de l'avion. Un avion est un système à part entière composé d'un certain nombre de sous-systèmes tels que l'équipage, le système de propulsion ou encore le système de contrôle de vol.

Le cycle de vie d'un système passe par un certain nombre de phases. L'ISO a principalement identifié les suivantes :

- La conception, qui va de l'identification des besoins à la proposition d'une solution réalisable
- Le développement, qui conduit à une première implémentation et validation du système
- La production, durant laquelle le système est produit en masse
- L'utilisation, consistant à la mise en œuvre du système pour répondre aux besoins des utilisateurs
- Le support, qui concerne l'installation, la maintenance et la logistique du système
- Le retrait, phase ou le système est détruit, archivé ou réutilisé

Ces différentes phases peuvent être organisées de différentes façons. Elles peuvent être effectuées de façon séquentielle, parallèle, incrémentale ou itérative. Chacun des sous-systèmes faisant partie d'un système plus complexe a son propre cycle de vie.

Chacune des phases du cycle de vie fait intervenir des processus. Un processus est défini comme un système d'activités qui utilise des ressources pour transformer des entrants en sortants. Une activité est un ensemble d'actions qui consomment du temps et des ressources et dont la réalisation est nécessaire pour qu'un système passe d'un état à un autre. L'ISO a ainsi identifié un certain nombre de types de processus (Figure 24).

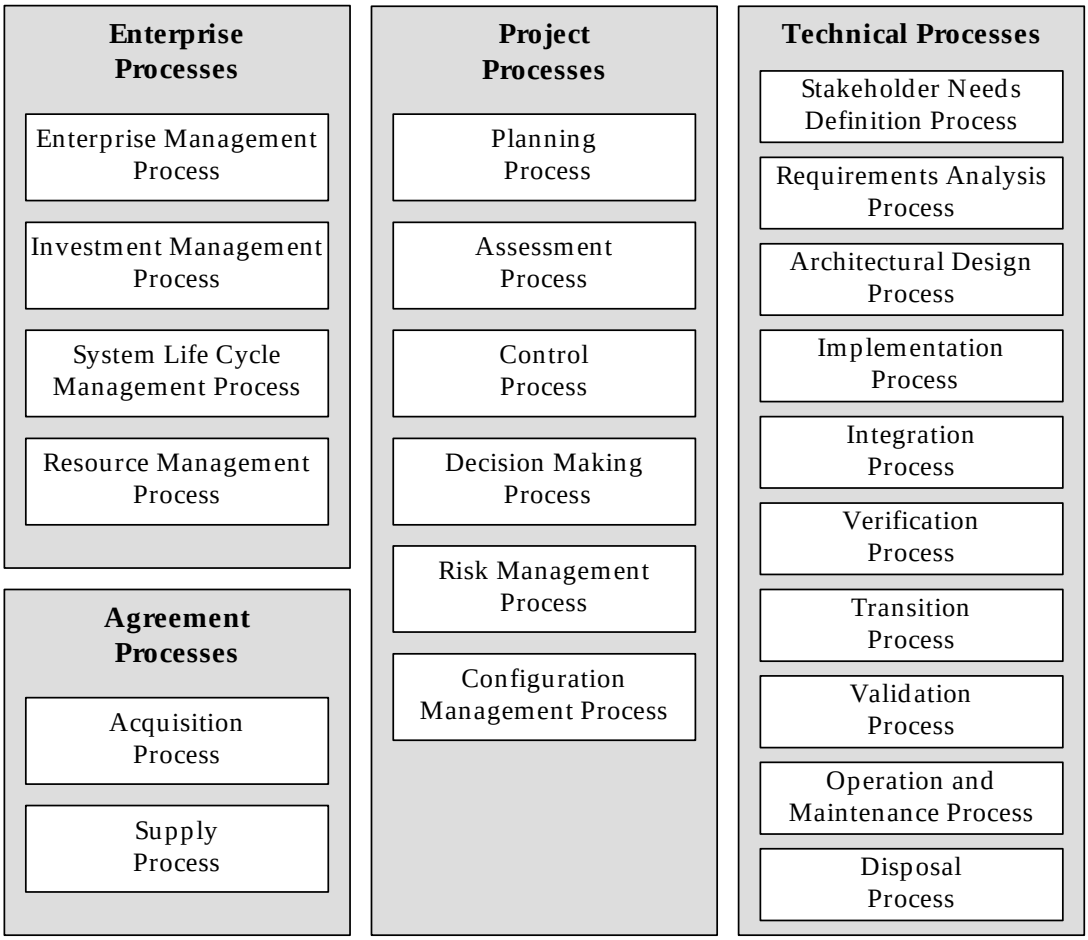


Figure 24. Typologie des processus définie par la norme ISO 15288

Les processus d'entreprise incluent tout ce qui concerne la définition des processus d'affaire, le contrôle de l'investissement dans de nouveaux projets, la gestion des ressources ainsi que la coordination entre projets. Les processus de contrat sont composés des processus d'acquisition et de fourniture de produits, de services et de systèmes. Les processus projet concernent les processus de gestion de projet : planification, contrôle, validation, gestion des risques, etc. Enfin, les processus techniques recouvrent l'ensemble des processus directement impliqués dans la production d'un système, depuis l'analyse des besoins, en passant par son intégration, jusqu'à son retrait. Tous ces processus peuvent être utilisés à toutes les phases du cycle de vie d'un système. La définition du cycle de vie d'un système consiste donc à décrire les phases qui le composent, ainsi que les différents processus qui sont mis en œuvre lors de chacune de ces phases.

Chaque système est sous la responsabilité d'un projet séparé. Une remarque importante faite dans la norme ISO/15288 est l'existence d'une correspondance structurelle entre la décomposition des systèmes et celle des projets (voir Figure 25). Il y a une corrélation entre le niveau de détail du système étudié et celui du projet. A chaque niveau, les processus pouvant être mis en œuvre par les projets sont les mêmes. Cette décomposition parallèle se retrouve jusqu'aux composants de base. Ainsi, la norme ISO/IEC 15288 prévoit-elle une relation avec la norme ISO/IEC 12207 dédiée à la définition des processus de développement de composants logiciels.

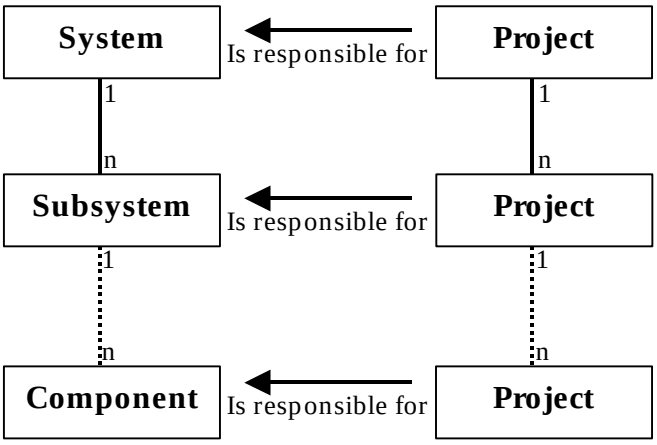


Figure 25. Parallèle de la décomposition entre systèmes et projets dans la norme ISO/IEC 15288

3.3. Dans le domaine du contrôle de gestion et de l'organisation

Philippe Lorino [52] identifie trois objectifs principaux pour le contrôle de gestion :

- Savoir : si l'on a atteint les objectifs que l'on s'était fixés
- Comprendre : pourquoi on les a ou on ne les a pas atteints
- Orienter l'action : pour améliorer ses performances

Les coûts sont une des mesures dont disposent les entreprises pour fixer leurs objectifs, et contrôler leur réalisation. En comptabilité de gestion, Philippe Lorino identifie deux grands types d'application :

- Les fonctions d'optimisation des coûts : qui consistent à trouver des voies permettant la réduction des coûts, notamment en améliorant la conception des produits, des processus et de l'organisation
- Les fonctions de maintenance des coûts : qui consistent à contrôler les coûts réels afin de s'assurer que ceux-ci ne s'écartent pas des normes

De nombreux concepts utilisés encore aujourd'hui dans les systèmes de gestion de coût proviennent du début du siècle, voire de la fin du siècle dernier. Ils ont été développés à partir des travaux de Taylor. La gestion taylorienne repose sur quatre principes de base :

- La stabilité : les systèmes, les savoirs et les mécanismes sont stables dans le temps
- L'information parfaite : les dirigeants disposent en permanence d'une information parfaite du système
- Augmenter la performance signifie réduire les coûts
- On peut identifier un facteur de coût déterminant pour chacun des produits

Aujourd'hui ces principes ne sont plus en cohérence avec la réalité. Les technologies, ainsi que les demandes du marché sont en constante évolution. Les systèmes sont de plus

en plus complexes, ce qui rend impossible toute vision exhaustive. Le coût n'est plus l'unique paramètre de performance à prendre en compte. Enfin, il est de plus en plus difficile d'identifier un facteur de coût dominant pour chaque produit. Les systèmes de gestion de coûts ont donc dû évoluer pour pallier les lacunes des systèmes classiques. C'est pour cela que l'approche basée sur les activités (ABC, Activity-Based Costing) a été développée et promue.

Une définition de l'ABC peut être trouvée dans le glossaire mis au point par le CAM-I, qui est un organisme international regroupant des grandes entreprises, des cabinets d'experts ainsi que des universitaires autour de problèmes communs aux entreprises, et qui essaie en particulier de poser les bases des systèmes de gestion modernes. L'ABC est une méthodologie qui mesure les coûts et les performances des activités, des ressources et des objets de coût, affecte les ressources aux activités et les activités aux objets de coût, et reconnaît une relation causale des inducteurs de coût vers les activités. Dans le modèle traditionnel, la consommation des ressources est directement affectée aux produits. Dans le modèle ABC, les produits consomment des activités, et ce sont ces activités qui consomment des ressources (Figure 26).

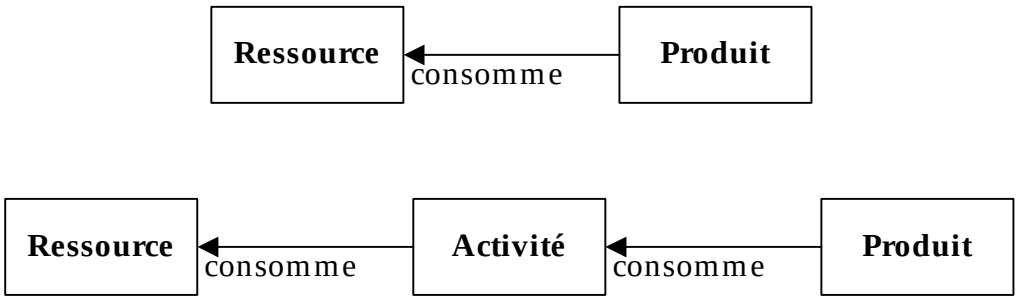


Figure 26. L'introduction du concept d'activité dans l'approche ABC

L'ajout du concept d'activité permet d'obtenir une précision plus importante dans l'analyse d'un système. En plus de l'activité, de nombreux travaux, pour la plupart français (Philippe Lorino, Pierre Mevellec [58]), intègrent également le processus. Cette brève présentation se base sur ceux-ci, tout comme le méta-modèle dédié à l'ABC que nous avons proposé dans [11].

Un processus est caractérisé par un résultat global unique destiné à un client identifié. Il a un facteur de déclenchement indiquant l'événement à l'initiative du processus (un même processus a potentiellement différents facteurs de déclenchement). Les processus peuvent ainsi être commandés (demande explicite du client, lancement d'un projet) ou autonomes (processus récurrent de maintenance, de suivi des clients). Les processus se décomposent en activités. Si l'on utilise une démarche « top-down », les processus constituent le point d'entrée permettant l'identification des activités. Dans le cadre d'une approche « bottom-up », les activités sont d'abord recensées, puis organisées en processus. Dans les deux cas il nous faut déterminer quelles activités sont partie prenante du processus, et quelles sont les règles qui spécifient leur ordonnancement (par exemple, le parcours d'une demande de crédit à l'intérieur des services d'une banque pourra être différent selon l'objet et le montant du crédit). Toute activité est située au sein d'une unité organisationnelle. Elle consomme des ressources pour produire un résultat. Parmi les activités composant un processus, certaines

sont considérées comme critiques, elles feront donc l'objet d'une attention accrue. Pour chaque activité on peut définir un coût et un délai. Enfin, les activités peuvent être typées : activité de conception, de réalisation ou de maintenance. Les ressources consommées et les valeurs produites au cours d'un processus peuvent être de natures totalement différentes : unité organisationnelle, matière première, matériel, logiciel, information, etc.

En plus de cette modélisation du système existant, l'ABC définit un certain nombre d'éléments dédiés au pilotage et à l'analyse. Les unités d'œuvre permettent de calculer la productivité d'une activité. Elles sont à choisir parmi les multiples mesures disponibles au niveau des ressources consommées et des produits générés. Pour chaque activité, on peut définir un niveau d'activité minimal, optimal et budgété. Ces unités d'oeuvre permettent l'analyse à posteriori, mais elles ne sont pas forcément adaptées pour le suivi des activités en cours. Pour cela, on définit des indicateurs de pilotage, qui permettent le contrôle de l'avancement des activités durant leur réalisation. Enfin, le dernier concept important dédié à l'analyse du système est l'inducteur de performance. Différents inducteurs peuvent être identifiés pour une même activité, chacun portant sur un aspect différent (coût, délai, qualité, etc.). L'inducteur peut être situé au sein même de l'activité, mais il est fréquent qu'il soit relié à une activité antérieure. De façon récursive on peut ainsi remonter dans l'arbre des causes afin d'identifier l'activité détenant l'inducteur primaire de performance. Les leviers d'action à disposition pour corriger les performances d'une activité sont donc spécifiés via ces inducteurs.

La méthode ABC apporte une précision accrue dans l'analyse des systèmes de production. Elle permet donc de répondre à des questions plus complexes, et touchant à des parties du système qui n'apparaissent pas dans les méthodes traditionnelles. On peut déterminer le coût de production et celui de non-production, on peut également isoler certaines parties de la chaîne de production pouvant être sous-traitées. Enfin, l'approche par les activités apporte au système une certaine pérennitée face aux changements organisationnels.

Cette approche a été reprise dans le domaine de l'organisation. La méthode ABM (Activity-Based Management) se concentre sur l'organisation des activités pour améliorer la valeur fournie au client, ainsi que le profit réalisé par l'entreprise (définition du CAM-I). La plupart des informations utilisées par l'approche ABM sont des résultats de la mise en œuvre de la méthode ABC.

3.4. Dans le domaine de la gestion de la qualité

La norme ISO 9000 est un standard incontournable dans le domaine de la gestion de la qualité. La définition normalisée de la qualité dans ISO 9000 se réfère à l'ensemble des caractéristiques d'un produit ou d'un service qui sont exigées par le client. La gestion de la qualité précise ce que l'organisation fait pour assurer que ses produits intègrent les exigences du client. La norme ISO 9000 recouvre en fait une famille de normes :

- ISO 9001 établit les exigences relatives à une organisation dont les activités vont de la conception et du développement à la production, à l'installation et aux prestations associées
- ISO 9002 concerne les organisations qui ne s'occupent pas de la conception et du développement; elle ne contient pas les exigences relatives à la maîtrise de la conception d'ISO 9001, les autres exigences étant identiques
- ISO 9003 s'adresse aux organisations dont les processus d'activité ne portent pas sur la maîtrise de la conception, la maîtrise des processus, les achats ou les prestations associées, et qui ont fondamentalement recours aux contrôles et aux essais pour assurer que les produits et les services finaux satisfont aux exigences spécifiées.

Un des principes de base de cette norme est l'approche processus [37]. Les avantages liés à cette approche sont présentés de la façon suivante : « un résultat escompté est atteint de façon plus efficiente lorsque les ressources et activités afférentes sont gérées comme un processus ». Les bénéfices attendus sont nombreux. Tout d'abord, une utilisation efficace des ressources permet de réduire les coûts et les délais. Les résultats peuvent être améliorés. Ils sont cohérents et prévisibles. Enfin, on peut se focaliser sur les points à améliorer et établir un ordre de priorité entre les différents processus.

Un processus est défini comme un ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie. Ces éléments peuvent être tangibles ou intangibles. Ils recouvrent l'ensemble des produits générés et des ressources consommées : composants, énergies, informations, etc. Les éléments d'entrée d'un processus sont généralement les éléments de sortie d'autres processus. C'est pourquoi, parmi les autres principes de base de la norme ISO 9000, on trouve également le management par l'approche système. Un des aspects de cette approche est la compréhension des interdépendances entre processus, et, parmi les avantages attendus, l'accent est mis sur l'intégration et l'alignement des processus.

Un certain nombre de familles de processus ont été identifiées :

- Les processus liés à la direction (définition des politiques et des objectifs qualité, communication)
- Les processus de management des ressources
- Les processus de réalisation du produit
- Les processus de mesure, d'analyse et d'amélioration,
- Les processus du système de gestion de la qualité

Un des apports majeurs des processus se situe au niveau de la mesure. On peut aussi bien mesurer la capacité du processus à produire le résultat escompté, que sa productivité. Ces mesures peuvent être faites avant la mise en œuvre du processus (prévisions), pendant, ou après. Elles peuvent être utilisées pour l'amélioration constante des processus. Pour cela, l'ISO conseille la démarche connue sous le nom de PDCA (Plan-Do-Check-Act) (Figure 27), soit en français : Planifier-Faire-Vérifier-Agir. Cette démarche est aussi bien applicable au

niveau d'un processus particulier que du système global (qui est constitué par des chaînes de processus).

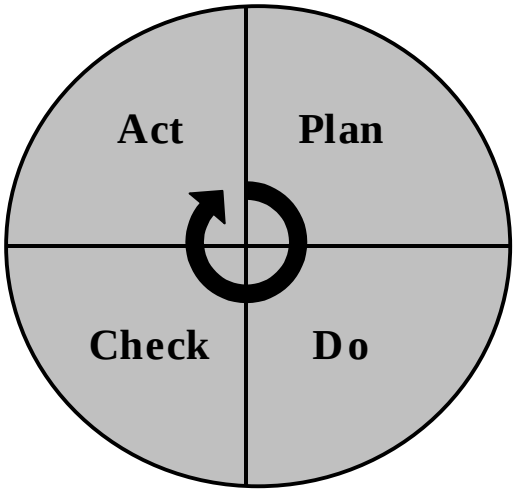


Figure 27. La démarche PDCA

Les quatre phases de cette démarche sont :

- Planifier : établir les objectifs et les processus nécessaires pour fournir des résultats correspondants aux exigences des clients et aux politiques de l'organisme
- Faire : mettre en œuvre les processus
- Vérifier : surveiller et mesurer les processus et le produit par rapport aux politiques, objectifs et exigences du produit et rendre compte des résultats
- Agir : entreprendre les actions pour améliorer en permanence les performances des processus.

La gestion des processus se fait au travers de documents tels que les descriptifs de processus pour tout ce qui concerne la planification et la maîtrise des processus, et d'enregistrements pour garantir que le processus a été mis en œuvre de manière conforme.

3.5. Dans le domaine du développement logiciel

Le processus de développement logiciel devient une préoccupation majeure dans le génie logiciel. Les organisations partie prenante dans le développement de logiciels ont pris conscience que la maîtrise des processus de production était un facteur clé pour assurer le succès d'un projet. Cela permet notamment de limiter les dépassements de coûts et délais et d'homogénéiser le niveau de qualité des produits développés.

Selon la définition qu'en donne Jean-Claude Derniame [20], un processus de développement logiciel inclut tous les éléments du monde réel impliqués dans le développement et la maintenance d'un produit, c'est-à-dire les artefacts, les supports de production, les activités, les agents et le support du processus. Carlo Montangero [20]

identifie les concepts de bases suivants (Figure 28) pour la description des processus de développement logiciels.

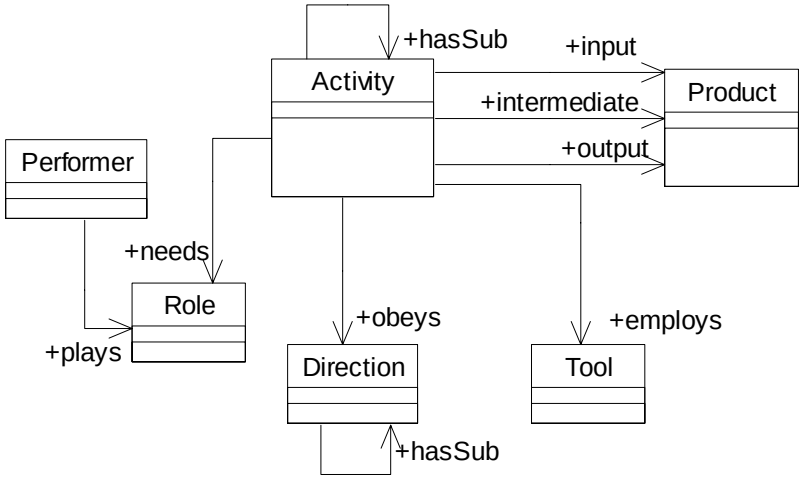


Figure 28. Les concepts de base de description d'un processus de développement logiciel

L'activité (*Activity*) est une étape du processus. Une activité produit des artefacts (*Product*). Elle met en oeuvre des rôles (*Role*) qui correspondent à des regroupements de responsabilités ou de compétences, qui seront affectés à des agents humains (*Performer*). Des agents logiciels (*Tool*) vont pouvoir être utilisés pour assister, voire automatiser, son exécution. Enfin, une activité est gouvernée par des règles, des procédures ou des normes (*Direction*).

On peut retrouver un ensemble plus ou moins complet de ces concepts à l'intérieur d'un certain nombre de méthodes. Parmi les plus connues se basant sur une approche objet, on peut citer OMT, Booch, OOSE ou encore Shlaer/Mellor. Une des caractéristiques de la plupart de ces méthodes est qu'elles présentent à la fois un processus et une notation. Ainsi la méthode OMT [82] définit un processus basé sur quatre grandes phases (analyse, conception du système, conception des objets, implémentation), ainsi qu'un méta-modèle objet. La mise au point d'UML a permis de séparer ces aspects produit et processus. L'objectif initial d'UML était la définition d'une méthode unifiant les trois principales propositions que sont OMT, Booch et OOSE. Devant la difficulté de la tâche, il a été décidé de viser un objectif moins ambitieux : la définition d'un formalisme unifié pour la description des produits utilisés ou élaborés dans un processus global de réalisation et de maintenance d'un logiciel à objets. La définition du processus a été prise en compte ultérieurement. Toutefois, à l'inverse de la notation, le processus n'a pas fait l'objet d'un consensus. La proposition de l'OMG, SPEM [65] (Software Process Engineering Metamodel) (voir la présentation de SPEM section suivante) n'est pas un processus, mais un méta-modèle de processus. L'avantage de SPEM est qu'ainsi les différentes méthodes pourront être définies dans un même formalisme, ce qui facilitera leur comparaison, voire leur intégration. Le RUP [45] (Rational Unified Process), produit commercial édité par la société

Rational, est un modèle de processus de développement logiciel s'appuyant sur UML et basé sur SPEM.

Le processus est également un indicateur de la maîtrise par les entreprises de leurs procédés de fabrication. Ainsi, le CMM [78] (Capability Maturity Model) se base sur l'analyse des processus de développement logiciel pour évaluer le degré de maturité des organisations. Cinq niveaux différents ont été identifiés (Figure 29). Au niveau initial (*Initial*) il n'y a pas de processus stable défini. Chaque projet définit (plus ou moins complètement) son organisation de façon ad hoc. Au niveau de maturité supérieur (*Repeatable*), les processus peuvent être répétés sur plusieurs projets. Un certain nombre de pratiques ont été identifiées et peuvent ainsi être mises en œuvre pour chaque nouveau développement. Au niveau 3 (*Defined*), un processus type est défini. Il peut être personnalisé selon les besoins rencontrés sur chacun des projets. Au niveau 4 (*Managed*), la qualité du processus et des produits résultats est évaluée en fonction de critères factuels. Enfin, au cinquième niveau (*Optimising*), ces critères sont pris en compte pour une amélioration continue du processus.

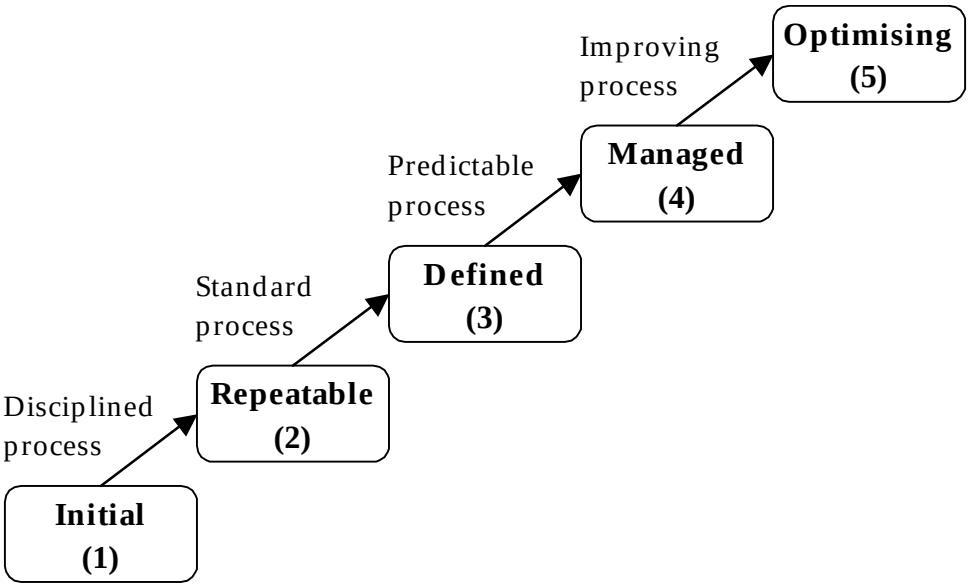


Figure 29. Les cinq niveaux définis par le CMM

3.6. Conclusion

Dans cette section, nous avons donné un aperçu de la définition du concept de processus et de son utilisation au travers de plusieurs domaines d'activité différents (ingénierie système, gestion et organisation, qualité, génie logiciel). On a pu voir que les diverses définitions contenaient tout de même des éléments communs. La décomposition d'un processus en activité, la consommation de ressources et la délivrance de produits par ces activités sont des schémas récurrents. Cela explique la facilité avec laquelle nous avons pu échanger avec des experts de chacun de ces domaines. De plus, les bénéfices attendus par l'approche processus sont également souvent similaires : gain de productivité, meilleure qualité du produit fini, pérennisation et diffusion des savoir-faire, etc.

Il y a différentes façons d'exprimer les similitudes qui viennent d'être dégagées. L'une d'entre elle consiste à les exprimer sous la forme de patterns. Ceci a été fait en partie par Jim Coplien [15]. Une telle approche est orientée vers la compréhension et la communication d'un savoir-faire. Notre objectif est plus opérationnel. Nous avons donc choisi l'option de la spécification explicite par méta-modèle qui vise à opérationnaliser autant qu'à comprendre.

4. Des formalismes de représentation de processus

4.1. Introduction

Il existe de nombreux travaux proposant des formalismes de représentation de processus. Chacun de ces formalismes définit des concepts qui lui sont propres, spécifiques au domaine ciblé ou à l'utilisation qui sera faite des spécifications. En effet, la modélisation de processus peut avoir des objectifs extrêmement variés. Bill Curtis et al. [18] en identifient cinq différents :

- Faciliter la compréhension humaine et la communication
- Assister l'optimisation
- Assister la gestion du processus
- Automatiser l'assistance à l'exécution
- Automatiser le contrôle de l'exécution

Il peut donc s'avérer intéressant de les comparer afin de discerner leurs points communs et leurs différences, et ainsi d'identifier des concepts génériques et d'autres spécifiques.

Une telle comparaison est cependant rendue difficile par la profusion des mécanismes de définition. Parmi les formalismes que nous allons présenter on trouve entre autre des ontologies, des schémas XML ou encore des méta-modèles. Pour permettre leur comparaison nous avons choisi de les représenter en utilisant un seul et même format, le MOF. Nous les avons donc tous décrits comme des méta-modèles basés sur le MOF. Même si nous avons essayé de respecter au maximum le formalisme initial, certaines divergences peuvent apparaître entre celui-ci et le méta-modèle que nous allons présenter. Cela tient au fait que chaque technique de définition présente des possibilités de description qui lui sont propres, et qui ne sont pas toujours offertes par les techniques de méta-modélisation.

Nous commençons par présenter les diagrammes de Gantt et les graphes PERT, deux techniques de planification de projet très utilisées et extrêmement simples. Puis, nous étudions les formalismes suivants :

- ARIS,
- BPML,

- CPR,
- ebXML,
- EDOC,
- IDEF3,
- PIF,
- PSL,
- SPEM,
- TOVE,
- UML,
- WPDL,
- WSFL,
- XLANG.

4.2. Les diagrammes de Gantt et les PERT

Les diagrammes de Gantt ont été créés par Henry Gantt dans les années 1920. Un diagramme de Gantt permet de décrire l'ensemble des activités d'un processus sous la forme de barres placées sur un calendrier. On a ainsi une vue graphique de l'ensemble des activités, de leurs durées et de leur ordonnancement.

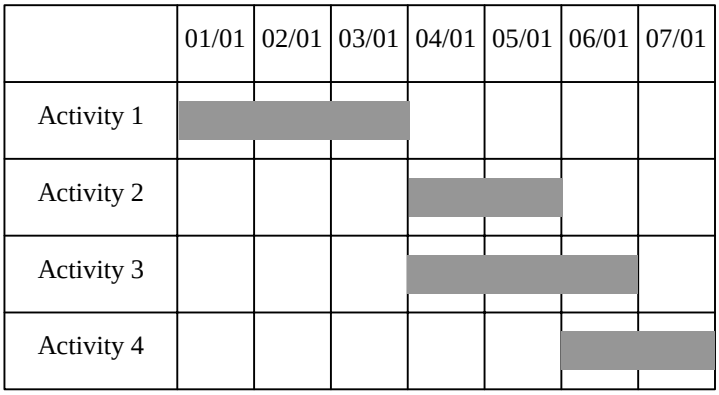


Figure 30. Exemple de diagramme de Gantt

A partir de l'exemple précédent (Figure 30) on peut en déduire le méta-modèle des diagrammes de Gantt. Il définit un processus comme un ensemble d'activités, chaque activité étant dotée d'une date de début et d'une date de fin (Figure 31). Cette version des diagrammes de Gantt est la plus simple. Il en existe de bien plus complexes, gérant des notions de pré-requis entre activités ou d'attribution de ressources. Les diagrammes de Gantt sont encore très utilisés, particulièrement pour la gestion de projet.

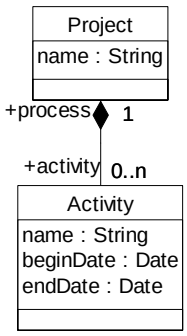


Figure 31. Méta-modèle d'un diagramme de Gantt simple

Les graphes PERT (Program Evaluation and Review Technique) ont tout d'abord été utilisés par le département américain de la défense. Un graphe PERT est un graphe orienté qui montre les activités, leur durée ainsi que leur ordonnancement (Figure 32). Un processus vu comme un graphe PERT est donc une suite d'activités, chaque activité ayant une durée et pouvant suivre ou précéder d'autres activités (Figure 33). Contrairement à l'activité d'un diagramme de Gantt, l'activité d'un graphe PERT ne définit pas de dates de début ou de fin.

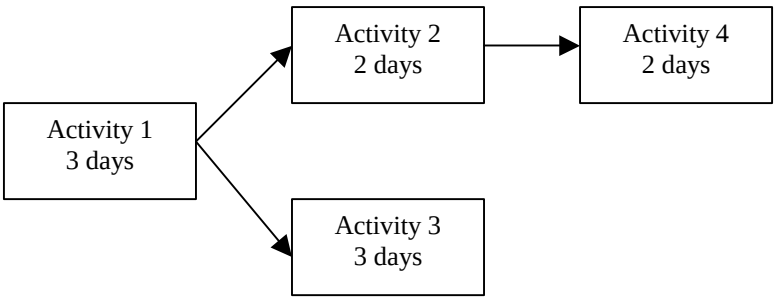


Figure 32. Exemple de PERT

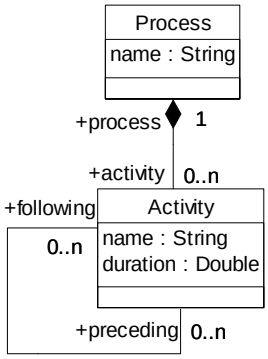


Figure 33. Méta-modèle d'un graphe PERT simple

Ces deux exemples simples montrent deux choses. D'une part les problématiques de représentation des processus sont relativement anciennes. D'autre part, même si un méta-modèle de processus est toujours constitué plus ou moins des mêmes entités (activité, temps), certaines options peuvent être prises qui permettent de se focaliser sur tel ou tel aspect et qui vont à terme générer des capacités différentes. Ainsi le diagramme de Gantt permet de connaître la date du jour auquel doit débuter et finir chacune des activités du processus, alors que le graphe PERT offre la possibilité de calculer le chemin critique. On

retrouve ainsi le principe énoncé préalablement : un modèle est caractérisé par l'ensemble des opérations que l'on peut lui appliquer (des questions que l'on peut lui poser). Ces opérations sont définies par le méta-modèle.

4.3. ARIS (Architecture of Integrated Information System)

ARIS [84] est une méthode de ré-ingénierie de processus mise au point par August-Wilhelm Scheer. Elle définit un formalisme de représentation des processus. ARIS est également un environnement de modélisation et de ré-ingénierie de processus métier édité par IDS Scheer. ARIS est divisé en cinq vues (voir Figure 34). Quatre d'entre elles sont indépendantes. Chacune de ces vues introduit un ensemble thématique d'entités. Ce sont :

- La Data view : les informations
- La Function view : les fonctions et leurs buts
- L'Organization view : les ressources humaines et matérielles
- L'Output view : les produits, qu'ils soient matériels ou immatériels (services).

La dernière vue (Control View) définit les relations entre les entités des quatre vues précédentes.

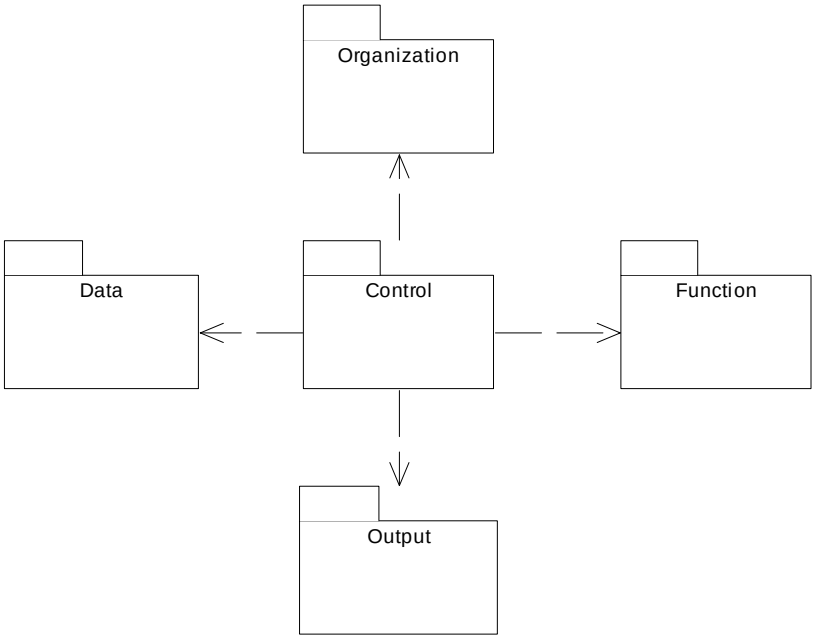


Figure 34. Architecture des vues d'ARIS

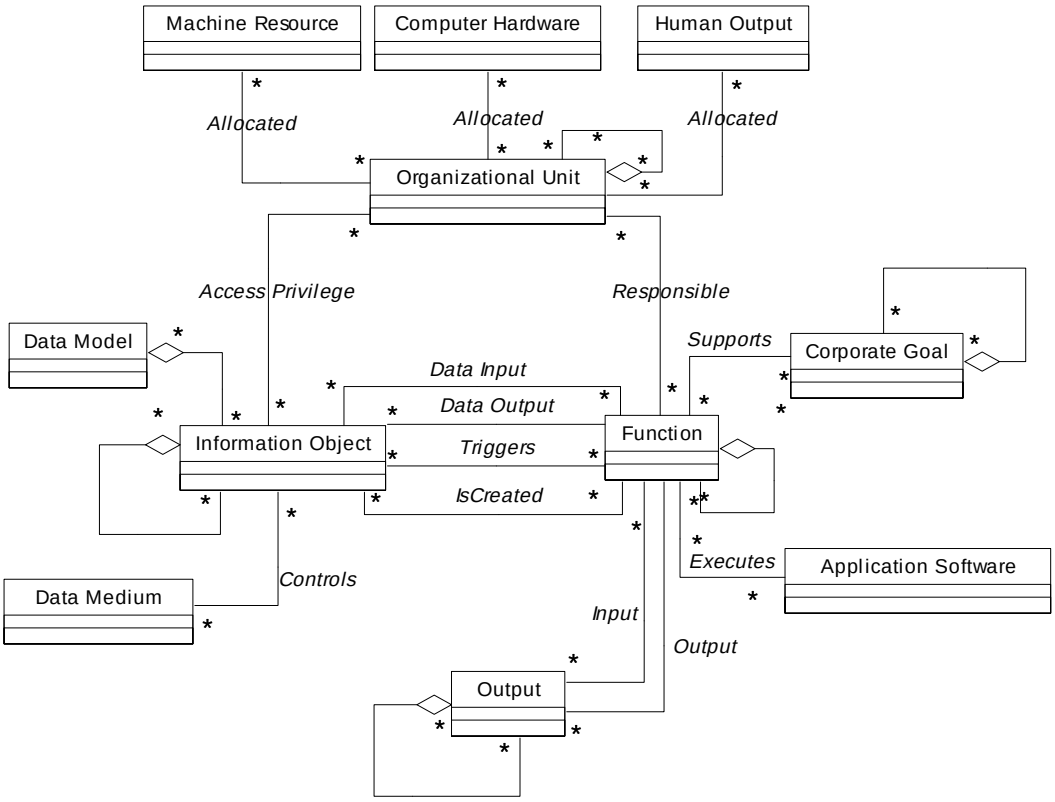


Figure 35. Noyau du méta-modèle d'ARIS

Pour ARIS un processus est un ensemble de fonctions (*Function*) cherchant à réaliser des objectifs (*Corporate Goals*) (voir Figure 35). Une fonction produit des résultats (*Output*) qui peuvent être aussi bien matériels qu'immatériels et traite des informations (*Information Object*). Elle est réalisée par des unité organisationnelles (*Organizational Units*) qui peuvent être des machines, des ordinateurs ou des ressources humaines.

L'organisation du méta-modèle d'ARIS est particulièrement intéressante puisqu'elle est basée sur quatre paquetages indépendants. Ces quatre paquetages sont ensuite mis en relation dans un cinquième. Les aspects données, organisation, fonction et services ont donc initialement été séparés dans des méta-modèles disjoints. Ces méta-modèles sont ensuite réunis pour donner une vision complète et intégrée de l'entreprise et de ses processus.

4.4. BPML (Business Process Management Language)

BPML [9] est une proposition du BPMI (Business Process Management Initiative), organisation regroupant diverses entreprises (Intalio, IBM, Rational, Mega International, Cap Gemini, etc.). Le BPMI s'est fixé comme objectif de définir un certain nombre de standards dans le domaine de la gestion de processus. Le premier d'entre eux, BPML, est un schéma

XML spécifiant un langage de modélisation de processus d'affaires exécutables par des BPMS (Business Process Management System), collaboratif et transactionnel.

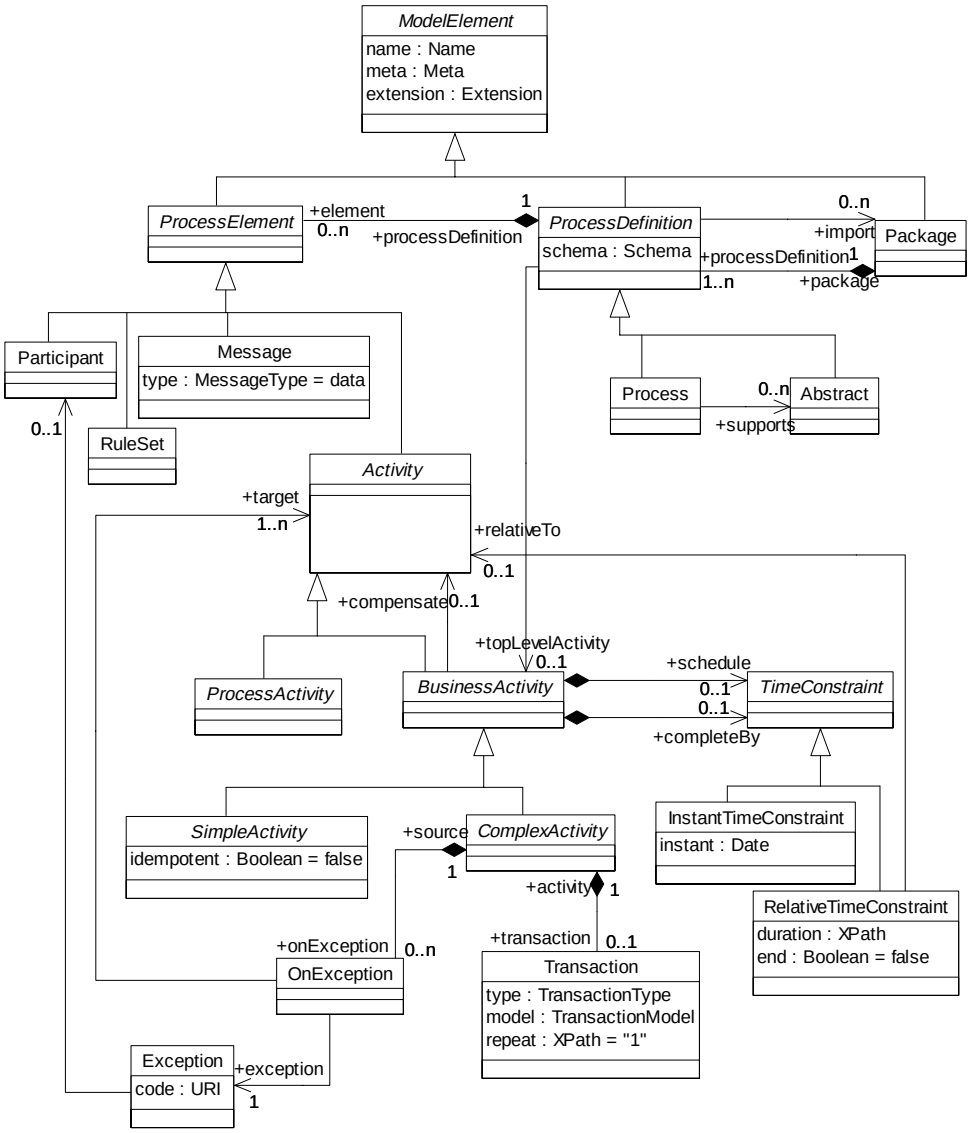


Figure 36. Concepts de base de BPML

BPML définit deux types de définition de processus : *Abstract* et *Process*. Un *Abstract* est un processus partiel, et donc non exécutable. Un *Process* définit un processus exécutable. Il peut implémenter plusieurs *Abstracts*. Les définitions de processus sont stockées dans des paquetages (*Package*) qui peuvent être réutilisés par la suite. Une définition de processus peut définir des messages (*Message*), des participants (*Participant*) et des ensembles de règles (*RuleSet*). Les messages définissent les informations qui transitent entre activités. Ils peuvent être de type *data*, *request* ou *response*. Les participants recouvrent l'ensemble des entités qui interagissent avec le processus (service Web, application, rôle organisationnel, etc.). Les ensembles de règles permettent de

regrouper les règles spécifiant les transitions orientant le flux de contrôle (activité *Switch*) ou l'acceptation ou non d'un message en entrée (*Input*).

Enfin, un processus définit une activité de plus haut niveau. Cette activité est soit une activité simple (*SimpleActivity*), soit une activité complexe (*ComplexActivity*). Elle peut avoir des contraintes de temps concernant son démarrage (*schedule*) ou sa fin (*completeBy*). Une activité complexe peut définir une transaction (*Transaction*). Dans le cas où une transaction est abandonnée, des activités de compensation (*compensate*) peuvent être lancées pour chacune des sous-activités de l'activité complexe. Enfin les activités complexes peuvent générer des exceptions (*Exception*). Une exception peut être adressée à un participant. Une activité peut être désignée pour traiter cette exception quand elle survient au cours de l'exécution d'une activité complexe (*OnException.target*).

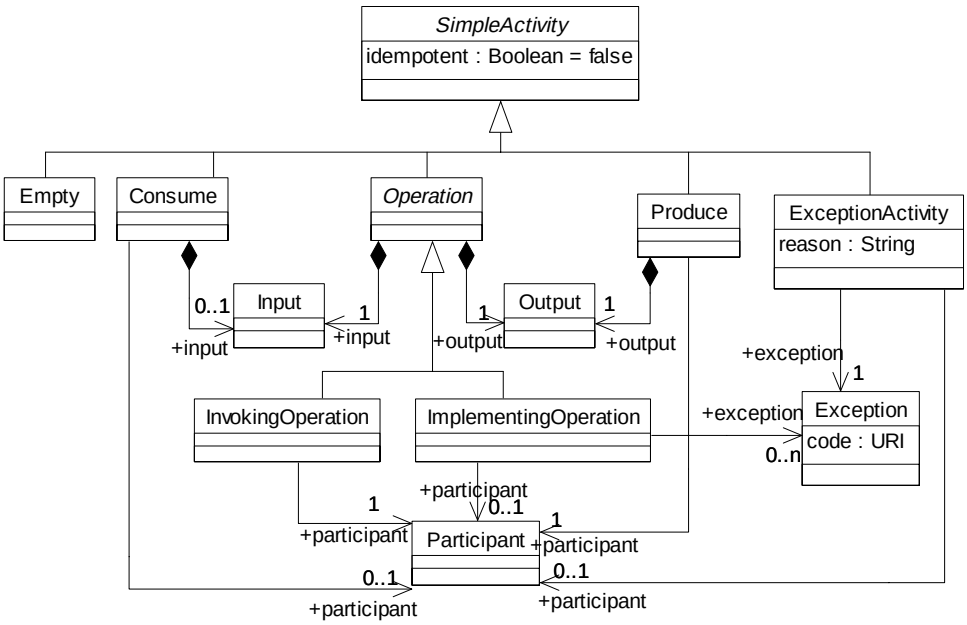


Figure 37. Activités simples de BPML

Les activités simples suivantes ont été identifiées :

- *Empty* : ne fait rien.
- *ExceptionActivity* : génère une exception.
- *Consume* : attend et consomme un message d'un participant.
- *Operation* : invocation d'une opération sur un participant (*InvokingOperation*), c'est-à-dire envoi du message sortant au participant et attente du message entrant, ou implémentation d'une opération (*ImplementingOperation*), c'est-à-dire consommation du message entrant et production du message sortant.
- *Produce* : produit un message et le délivre à un participant.

La plupart de ces activités peuvent donner lieu à des échanges d'information (*IO*), représentés par les concepts d'*Input* et d'*Output*. Un *Input* accepte un message destiné à un processus. Des *Assignments* permettent d'intégrer les informations contenues par le

message dans le contexte du processus. Cette affectation peut être contrainte par des règles (*Rule*), exprimée soit comme des conditions (*ConditionRule*), c'est-à-dire des expressions booléennes XPath, soit par des *RuleSets*. Un *Output* construit un message délivré par le processus. Des *Assignments* permettent de préciser comment les différents champs du message peuvent être valués à partir des données du processus. Ce concept de donnée de processus n'est pas précisément défini dans la version actuelle de la spécification. Il semble que chaque processus soit pourvu d'un contexte où il peut entreposer et rechercher des informations grâce à des expressions XPath. Les versions futures de BPML devraient clarifier ce point.

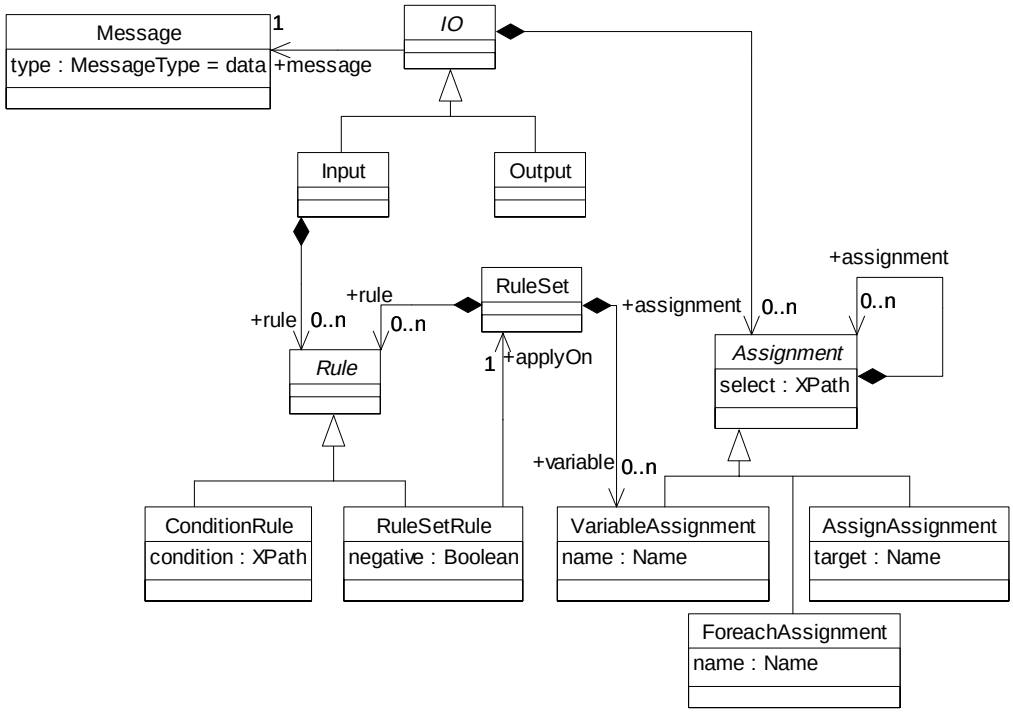


Figure 38. Les échanges de données dans BPML

BPML définit les activités complexes suivantes :

- *All* : une activité qui est terminée lorsque toutes ses sous-activités sont terminées. Ces sous-activités sont effectuées en parallèle.
- *Choice* : une activité qui est terminée dès que l'une de ses sous-activités est terminée.
- *Foreach* : une activité qui répète une séquence d'activités sur un ensemble de valeur.
- *Sequence* : une activité qui est terminée lorsque toutes ses sous-activités sont terminées. Ces sous-activités sont effectuées en séquence.
- *Switch* : une activité qui est terminée lorsque toutes ses sous-activités candidates à l'exécution sont terminées. Si aucune activité n'est candidate, et si un lien *otherwise* est défini, l'activité reliée par ce lien est exécutée.

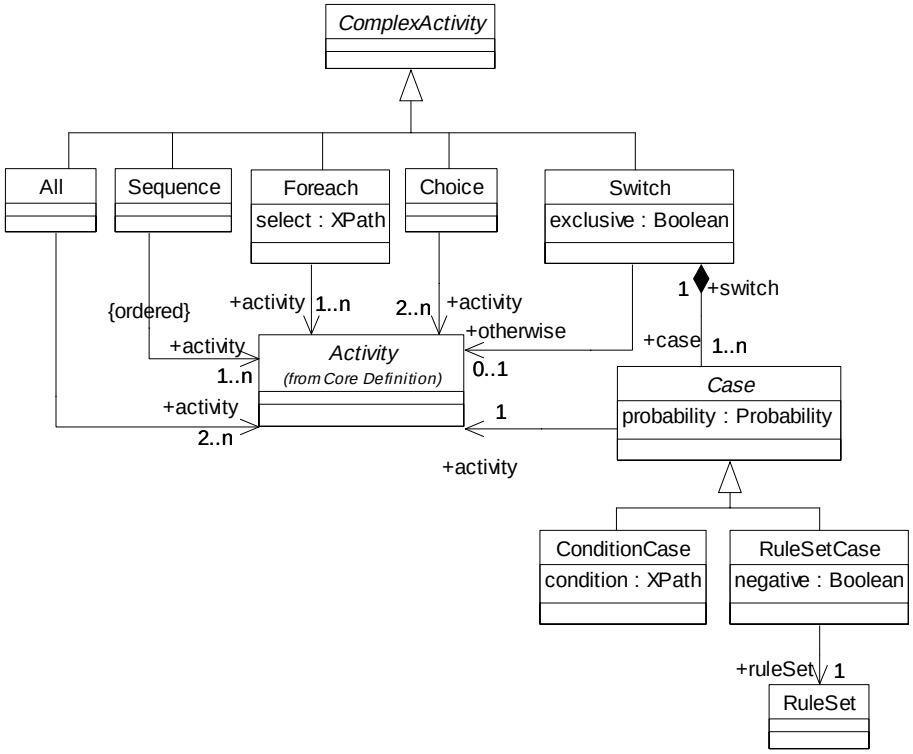


Figure 39. Activités complexes de BPML

Enfin, le dernier type d'activité, les activités procédurales, sont :

- *Assign* : exécute une affectation.
- *Complete* : termine le processus.
- *Join* : attend la fin de l'exécution d'un sous-processus.
- *Release* : libère une donnée de la valeur qui lui a été affectée.
- *Repeat* : répète l'activité parente.
- *Spawn* : démarre un sous-processus.

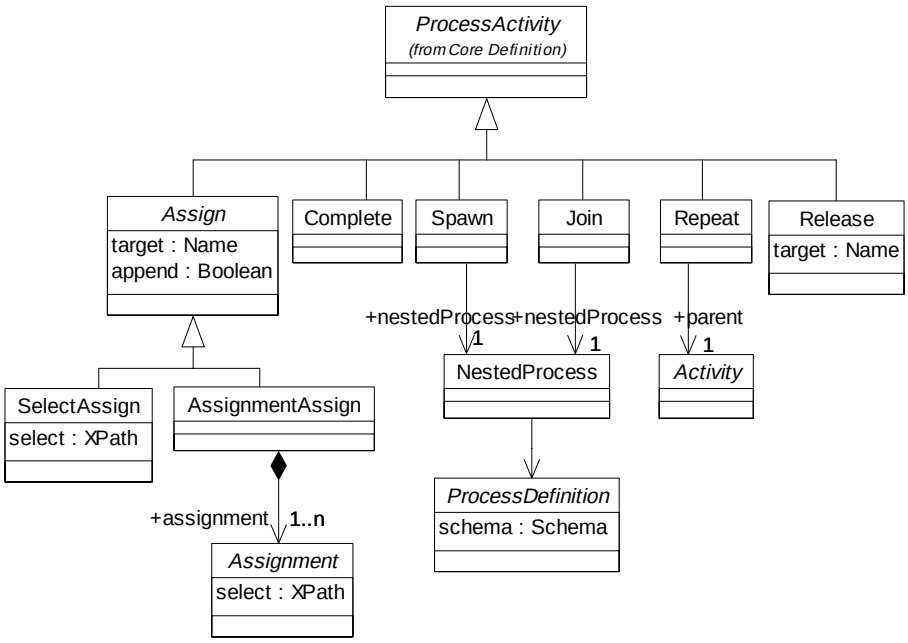


Figure 40. Activités processus de BPML

4.5. CPR (Core Plan Representation)

CPR [80] est un projet du DARPA (Defense Advanced Research Project Agency) qui se concentre principalement sur la planification (spécification d'une liste d'actions ayant pour but de répondre à un ensemble d'objectifs) ainsi que sur la prévision (spécification des moments auxquels seront réalisées les activités et des quantités de ressources utilisées).

Le formalisme défini par CPR est extensible. De nouvelles entités peuvent être créées par spécialisation des entités de base. CPR peut ainsi être adapté à un domaine donné. De telles extensions ont déjà été réalisées dans le domaine militaire.

CPR a pour but de modéliser un plan, c'est-à-dire un ensemble d'actions réalisées pour répondre à des objectifs. Les concepts de CPR sont les actions (*Action*), les acteurs (*Actor*), les objectifs (*Objective*) et les ressources (*Resource*) (Figure 41). Une action est réalisée par un acteur pour accomplir des objectifs. Des ressources peuvent être consommées lors de la réalisation d'une action. L'acteur d'une action peut être la ressource d'une autre.

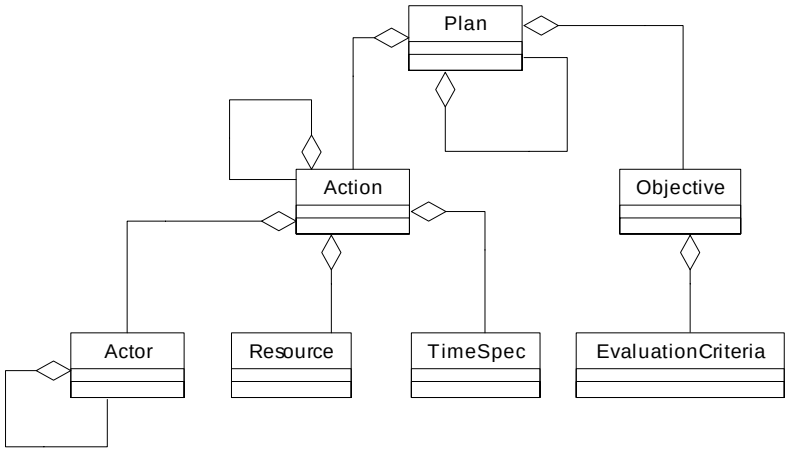


Figure 41. Noyau du méta-modèle de CPR

Le méta-modèle présenté précédemment (Figure 41) est prévisionnel et ne permet pas de mettre en relation un modèle de plan avec ses occurrences. C'est dans ce but qu'a été ajouté le concept de *WorldModel* qui permet de représenter des instances de plan (Figure 42). L'exécution d'un plan est structurée de la même manière que sa conception mais alors qu'une conception prévoit la façon dont doivent se dérouler les occurrences, l'exécution du plan représente ce qui s'est effectivement passé.

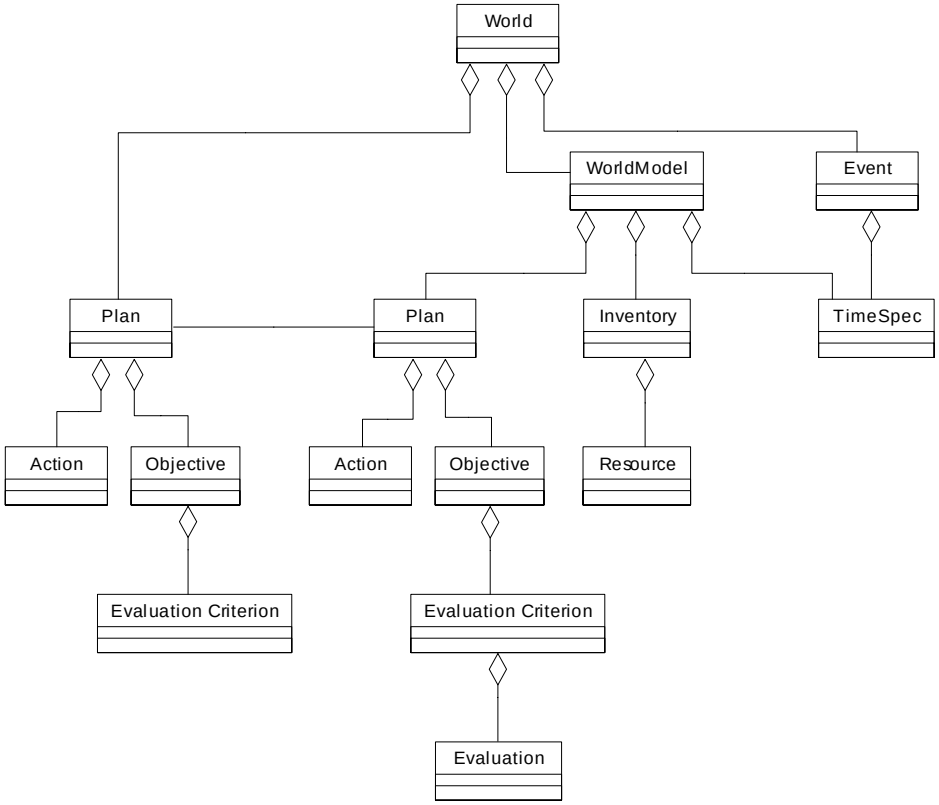


Figure 42. Aspects statiques et dynamiques de CPR

On a donc deux aspects des processus qui cohabitent à l'intérieur d'un même méta-modèle. Si on fait le parallèle avec la boucle PDCA utilisée dans le domaine de la gestion de la qualité, on peut voir apparaître dans ce méta-modèle la partie planification (P), la partie mise en œuvre (D) et la partie mesure (C) qui est définie par le concept d'*Evaluation*. Par contre le volet action (A) n'est pas représenté. Pour l'introduire, il faudrait pouvoir établir des liens entre deux versions de plan de conception, et expliciter les variations entre ces deux versions ainsi que les éléments ayant entraîné ces modifications.

4.6. ebXML

ebXML [24] est une proposition conjointe de l'UN/CEFACT (organisme des Nations Unies) et d'Oasis. Il se base sur des travaux antérieurs réalisés au sein de l'UN/CEFACT, principalement autour d'UMM (UN/Cefact Modeling Methodology). UMM proposait un méta-modèle pour la description des processus d'affaire. Cette partie a été reprise et enrichie dans ebXML. Ce dernier vise à la définition des collaborations entre partenaires au cours des processus d'affaires au travers de transactions et d'échanges de documents. Le formalisme résultat est un schéma XML (dernière version en date de mai 2001), mais les concepts manipulés sont explicités sous la forme d'un modèle UML à titre informatif.

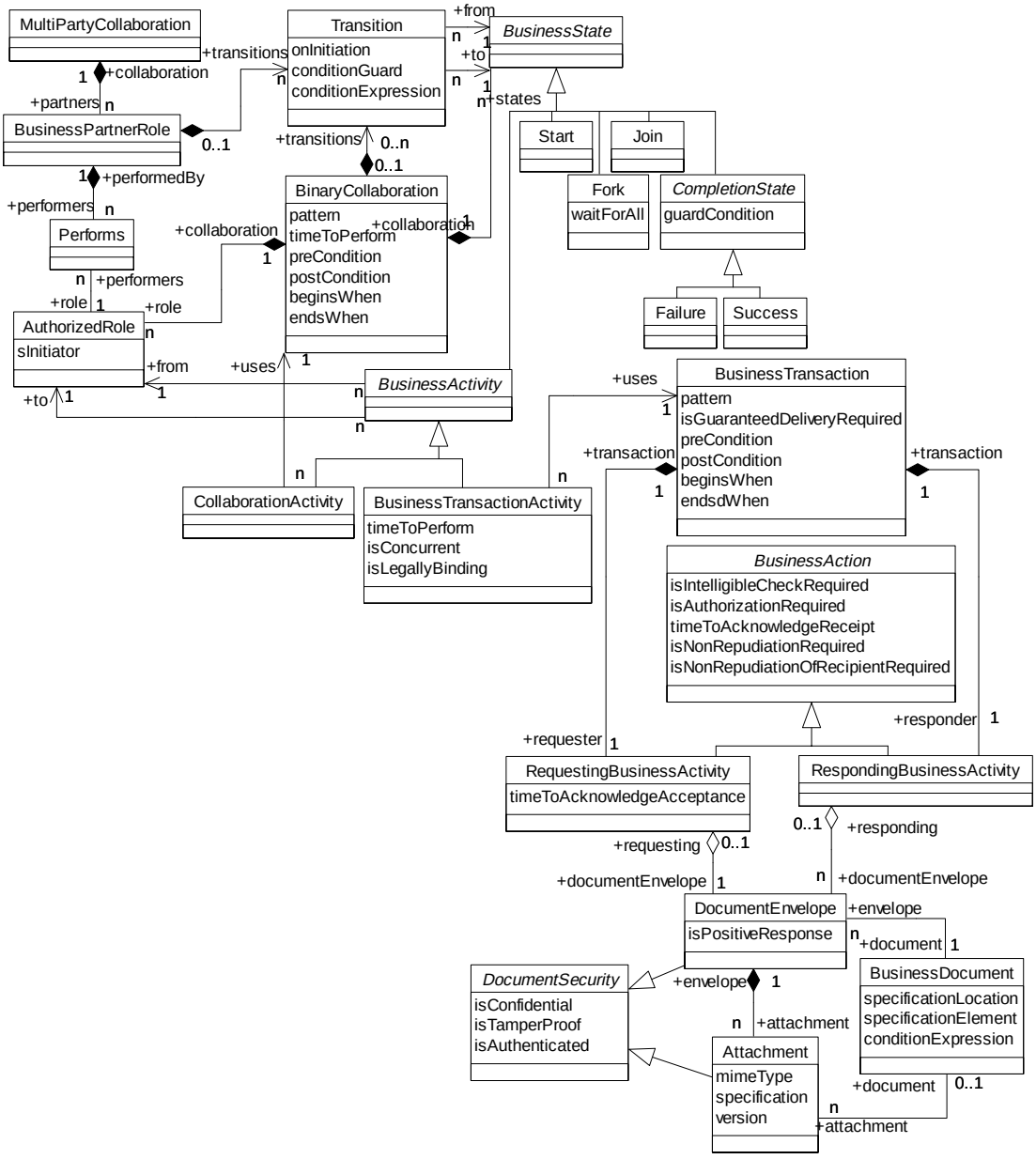


Figure 43. Méta-modèle de ebXML

Les processus ebXML sont vus comme des collaborations multipartites (*MultiPartyCollaboration*). Une collaboration multipartite implique différents partenaires (*BusinessProcessPartnerRole*). Ceux-ci interagissent au travers de collaborations bipartites (*BinaryCollaboration*), dans lesquelles deux partenaires vont jouer un rôle parmi les rôles autorisés (*AuthorizedRole*). Une collaboration multipartite est donc ramenée à un ensemble de collaborations bipartites. Une collaboration bipartite passe par un certain nombre d'états (*State*) ordonnancés par des transitions (*Transition*). Ces transitions sont soit définies par la collaboration, soit par un partenaire. Cela permet à tout partenaire de personnaliser ses processus. Ainsi, dans une collaboration de type achat/vente entre un acheteur et un vendeur, les interactions entre les deux parties peuvent être similaires, alors que le

processus côté vendeur peut être différent selon que celui-ci soit producteur ou simple revendeur de l'article.

Parmi les états ont été identifiés le point de départ de la collaboration (*Start*), les points finaux (*CompletionState*), permettant de représenter les cas de succès (*Success*) ou d'échec (*Failure*), et des points de parallélisme (*Fork*) et de synchronisation (*Join*). Enfin ces états peuvent également être des activités d'affaire (*BusinessActivity*) qui définissent soit une sous-collaboration (*CollaborationActivity*), soit une activité transactionnelle (*BusinessTransactionActivity*), entre deux rôles, l'initiateur (*from*) et le répondeur (*to*). L'activité transactionnelle est atomique. Elle décrit un échange de document entre deux rôles, échange réalisé au sein d'une transaction (*BusinessTransaction*) et décomposé en deux parties : la requête (*RequestingBusinessActivity*) et la réponse (*RespondingBusinessActivity*). La requête est obligatoire, par contre la réponse est optionnelle (il peut arriver que la requête définisse seulement un transfert de documents sans attente de réponse).

Un flux de document est représenté comme une enveloppe (*DocumentEnvelope*) contenant un document principal (*BusinessDocument*). En plus de celui-ci, un certain nombre de documents optionnels peuvent être attachés (*Attachment*). Des niveaux de sécurité, définis dans *DocumentSecurity*, peuvent être affectés à l'enveloppe ainsi qu'à chacun des documents attachés.

Une originalité d'ebXML est qu'il se concentre sur les transactions entre utilisateurs. Les actions de plus bas niveau sont des échanges de documents entre partenaires. Ce méta-modèle est donc surtout dédié pour la description des processus inter-entreprises. Par contre, il n'est pas adapté à la description de processus entièrement interne à une entité. D'ailleurs, le terme collaboration est utilisé de préférence à processus.

4.7. EDOC (Enterprise Distributed Object Computing)

L'objectif d'EDOC [72] est la représentation des composants constituant un système, et leur implication dans les processus métier, c'est-à-dire les événements auxquels ils réagissent et les règles métier contraignant leurs actions. EDOC est défini comme un méta-modèle et comme un profil UML. Des correspondances ont été définies entre les concepts du méta-modèle et les entités du profil. Nous nous intéressons ici plus particulièrement à la description d'EDOC sous la forme d'un méta-modèle. Ce dernier est divisé en trois grandes parties :

- L'ECA (Enterprise Collaboration Architecture) qui définit les composants, processus, événements et règles métier indépendamment de la plate-forme d'exécution.
- Le méta-modèle de patterns, qui permet de décrire des savoir-faire.
- Des méta-modèles techniques. En particulier la spécification décrit un méta-modèle pour Java, les EJB (Enterprise Java Bean), ainsi que les FCM (Flow

Composition Model). FCM est un formalisme utilisé au sein d'outils comme MQ-Series pour la description des interactions et du flot d'information entre composants.

L'ECA est divisé en cinq parties distinctes interdépendantes. Chacune est articulée autour d'un concept central (composant, objets métier, événements, processus et relations). Nous nous intéressons ici principalement à l'ECA, et plus particulièrement à la partie concernant la définition des processus. Nous ne présentons donc pas une vue exhaustive d'EDOC. La figure suivante (Figure 44) décrit les concepts sur lesquels se base la définition de processus. Afin de faciliter la compréhension, nous avons simplifié le méta-modèle de départ. En particulier, nous avons supprimé plusieurs niveaux d'héritage, ne laissant apparaître que les concepts et relations que nous jugions les plus pertinents.

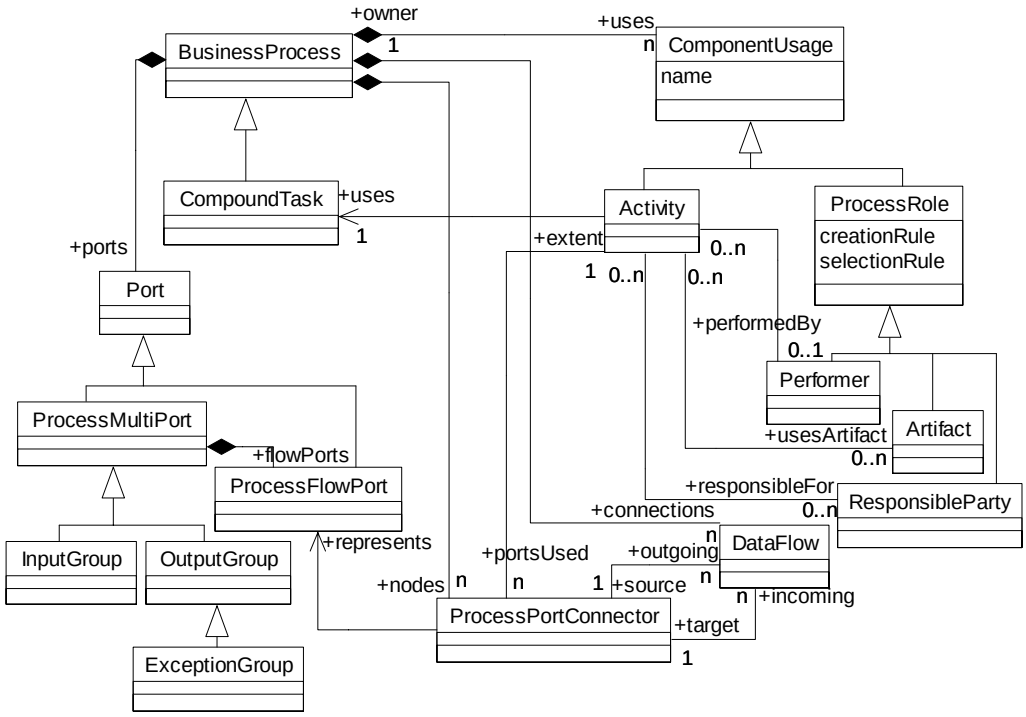


Figure 44. Définition des processus dans EDOC

Un *ProcessComponent* décrit le contrat d'un composant actif, c'est-à-dire qui réalise des traitements. Les processus (*BusinessProcess*) et les tâches composées (*CompoundTask*) sont des *ProcessComponents*, tout comme les gestionnaires de données (*DataManager*) ou les entités métier (*Entity*) que nous ne présentons pas ici.

Le contrat d'un composant est exprimé en terme de ports (*Port*) qui indiquent les points de connexion permettant les collaborations inter-composants. Le contrat d'une tâche composée est uniquement constitué de *ProcessMultiPort*. Les *ProcessMultiPort* sont composés de *ProcessFlowPort*. Les *InputGroups* décrivent des ports d'entrée, les *OutputGroups* des ports de sortie et les *ExceptionGroups* des ports d'erreur.

Un processus, tout comme une tâche composée, est une succession de communication entre composants via leurs ports. Ces composants sont identifiés par les activités (*Activity*),

dont le diagramme d'état est présenté Figure 45. Une activité peut soit invoquer une tâche composée soit être réalisée par un *Performer*. Une activité peut également utiliser des ressources (*Artifact*) et avoir des responsables (*ResponsibleParty*). *Performer*, *Artifact* et *ResponsibleParty* recouvrent des connexions vers des composants. La propriété *selectionRule* du *ProcessRole* permet de spécifier l'ensemble des composants pouvant être utilisés. La propriété *creationRule* décrit la façon dont le composant peut être créé.

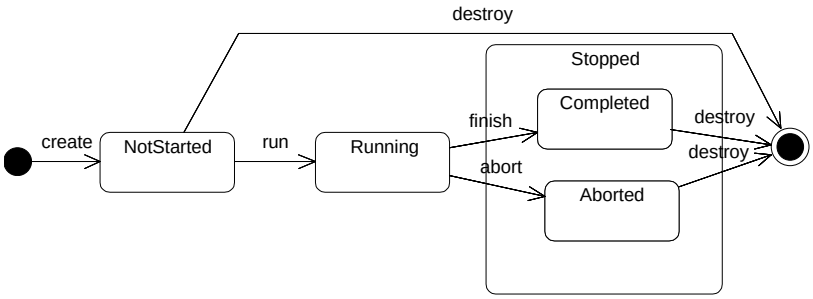


Figure 45. Diagramme d'état d'une activité dans EDOC

Enfin, l'ordonnancement entre activités est défini par les flots de données (*DataFlow*) qui transitent entre connecteurs (*ProcessPortConnector*). Ces connecteurs sont utilisés par une activité. L'ordonnancement entre connecteurs détermine donc l'ordonnancement des activités.

La définition d'EDOC a été influencée par le MDA. En effet, on observe bien la séparation entre PIM et PSM. Une deuxième partie d'EDOC devrait définir des correspondances entre l'ECA et les méta-modèles techniques.

4.8. IDEF3 (Integration Definition)

IDEF3 [56] est un formalisme défini par la société KBSI, en collaboration avec quelques universités, destiné à la capture de séquences d'activités. La version présentée ici date de 1995, et semble être la dernière en date. IDEF3 est avant tout un formalisme graphique. Toutefois, un langage formel basé sur KIF (Knowledge Interchange Format) est joint en annexe de la spécification. KBSI a développé deux outils basés sur IDEF3 :

- ProCap qui permet la définition de processus
- ProSim qui intègre en plus des fonctionnalités de ProCap un module de simulation

IDEF3 peut être découpé selon deux perspectives : la perspective processus (Figure 46) et la perspective objet (Figure 47). Ces deux perspectives ne sont pas totalement indépendantes. Elles peuvent se référencer l'une l'autre (Figure 48).

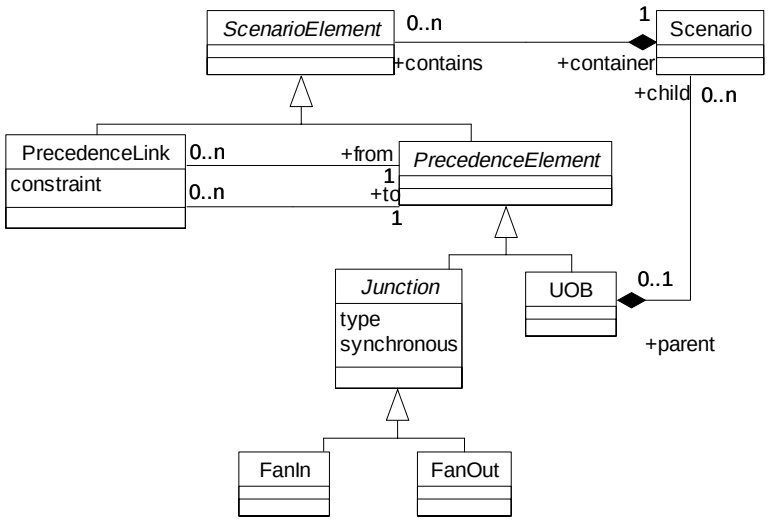


Figure 46. La perspective processus d'IDEF3

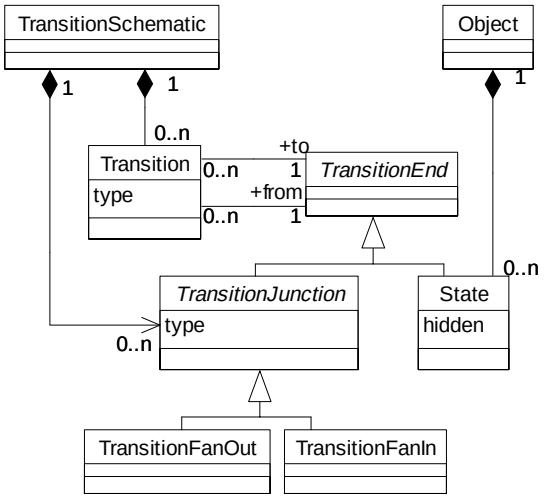


Figure 47. La perspective objetd'IDEF3

La perspective processus permet de décrire des scénarios (*Scenario*). Un scénario est un enchaînement d'unités de comportement (*UOB*), de points de jonction (*Junction*) et de références (*Referent*). Cet enchaînement est défini par des liens de précédence (*PrecedenceLink*). Les liens de précédence peuvent définir des contraintes. Ainsi on peut définir qu'un *UOB* B doit toujours être précédé d'un *UOB* A. Les *UOB* (Unit Of Behaviour) représentent des activités. Généralement, leur nom est un verbe suivi d'un complément (*Prendre une décision*, *Planifier le processus*). Un UOB peut être décomposé en un sous-scénario, voire même en plusieurs sous-scénarios afin de définir des vues différenciées. Les jonctions (*Junction*) permettent de définir des points de convergence (type *AND*) ou de divergence (types *OR* et *XOR*), que ce soit sur le flux entrant (*FanIn*) ou sortant (*FanOut*).

La perspective objet permet de décrire les objets et leurs différents états. Un objet (*Object*) peut passer par différents états (*State*). Ces états peuvent être ordonnancés dans des diagrammes de transition (*TransitionSchematic*) à l'aide de transitions (*Transition*) et de

jonctions (*TransitionJunction*) permettent de définir des points de convergence ou de divergence, que ce soit sur le flux entrant (*TransitionFanIn*) ou sortant (*TransitionFanOut*).

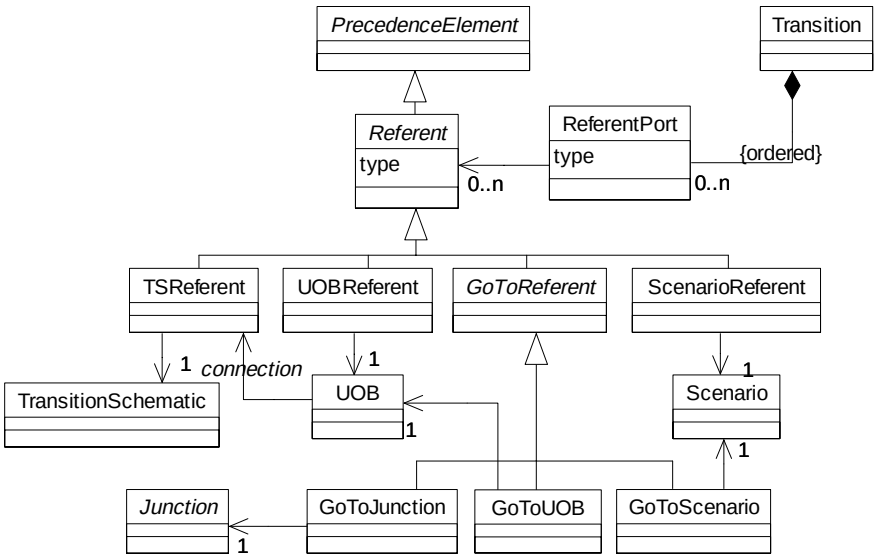


Figure 48. Le mécanisme de référence

Les références permettent de réutiliser des éléments définis par ailleurs dans un scénario ou dans un diagramme de transition. Dans un scénario, on peut intégrer des références vers des UOB ou d'autres scénarios ainsi que des références de type GOTO. Les deux premières signifient qu'on insère l'UOB ou le scénario sélectionné dans le flux d'activité. La référence de type GOTO va permettre de définir des boucles ou des sauts dans le scénario. Une référence vers un diagramme de transition ne peut pas être reliée par des liens de précédence. Par contre elle peut être connectée à un UOB, indiquant que le diagramme de transaction sera initié au cours de l'exécution de l'UOB.

Dans un diagramme de transition, on peut intégrer des références vers des UOB, des scénarios ou d'autres diagrammes de transition. Ces références sont reliées à des transitions. Elles indiquent les événements qui provoqueront le passage d'un état à l'autre. Un événement est le début de l'exécution d'un UOB ou d'un scénario ou l'initiation d'un diagramme de transition. Leur ordre de réception peut être contraint. Deux événements peuvent également être simultanés ou avoir un ordre indéfini (attribut *type* de l'entité *ReferentPort*).

Une des spécificités d'IDEF3, que l'on retrouve dans des propositions plus récentes comme UML, est d'établir des relations entre les activités des processus et les états des objets. C'est la même idée qu'on retrouve dans SADT, où l'on peut décomposer un système selon deux vues complémentaires : les actigrammes et les datagrammes. IDEF3 établit donc un lien fort entre les processus et les produits, les processus modifiant les produits et les produits contraignant les processus.

4.9. PIF (Process Interchange Format)

PIF ([46]) est issu des besoins de diverses organisations (MIT, DEC, Stanford, l'université de Toronto, l'université d'Hawaii, l'université d'Edimbourg, KBSI) de partager leurs modèles de processus. Les spécifications de PIF définissent à la fois un méta-modèle de processus et une syntaxe basée sur KIF. Le projet PIF a débuté en octobre 1993 et la notation a progressivement évolué au cours des années. Aujourd'hui PSL (voir 4.10), formalisme plus orienté vers la représentation des processus de production, et PIF ont fusionné pour intégrer des concepts de processus tertiaires et industriels dans une seule et unique ontologie.

PIF est basé sur un ensemble de concepts non minimal, permettant l'expression de processus simples. Cet noyau peut être enrichi en utilisant le mécanisme d'extension appelé Partially Shared View (PSV). Un module PSV étend le module racine (le noyau du méta-modèle) ou un autre module PSV (Figure 49). Ce nouveau module PSV définit de nouvelles entités en spécialisant des entités déjà définies dans des modules de plus haut niveau.

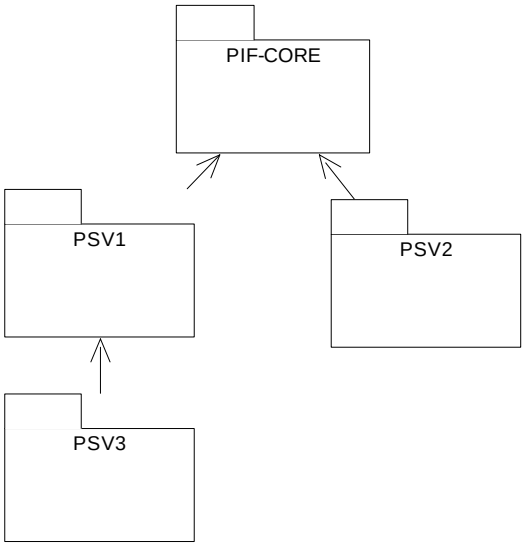


Figure 49. Mécanisme des Partially Shared View

PIF définit un processus comme un ensemble d'activités qui ont des relations entre elles ainsi qu'avec des objets à des moments dans le temps. Tous ces concepts héritent d'une entité commune nommée *Entity* (Figure 50).

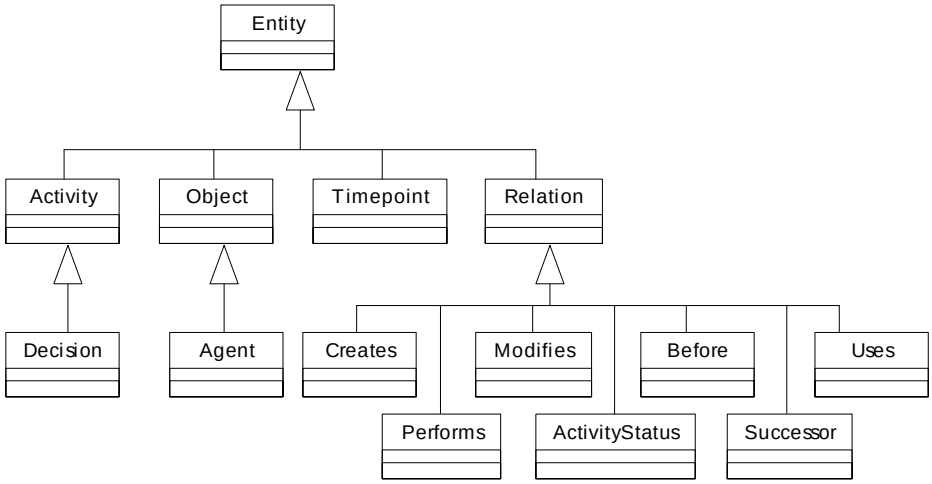


Figure 50. Hiérarchie des entités de PIF

Le concept d'*Activity* définit tout ce qui arrive dans le temps, que ce soit un processus, une tâche ou même un événement. Les *Timepoints* peuvent être soit des dates précises soit des instants indéfinis (par exemple l'instant auquel une tâche prend fin, où un événement survient). Les *Objects* représentent toutes les autres entités impliquées dans un processus. Cette notion recouvre les artifacts, les données, les outils, ou même les acteurs humains ou mécaniques (*Agent*). Toutes ces entités sont reliées par des relations (Figure 51).

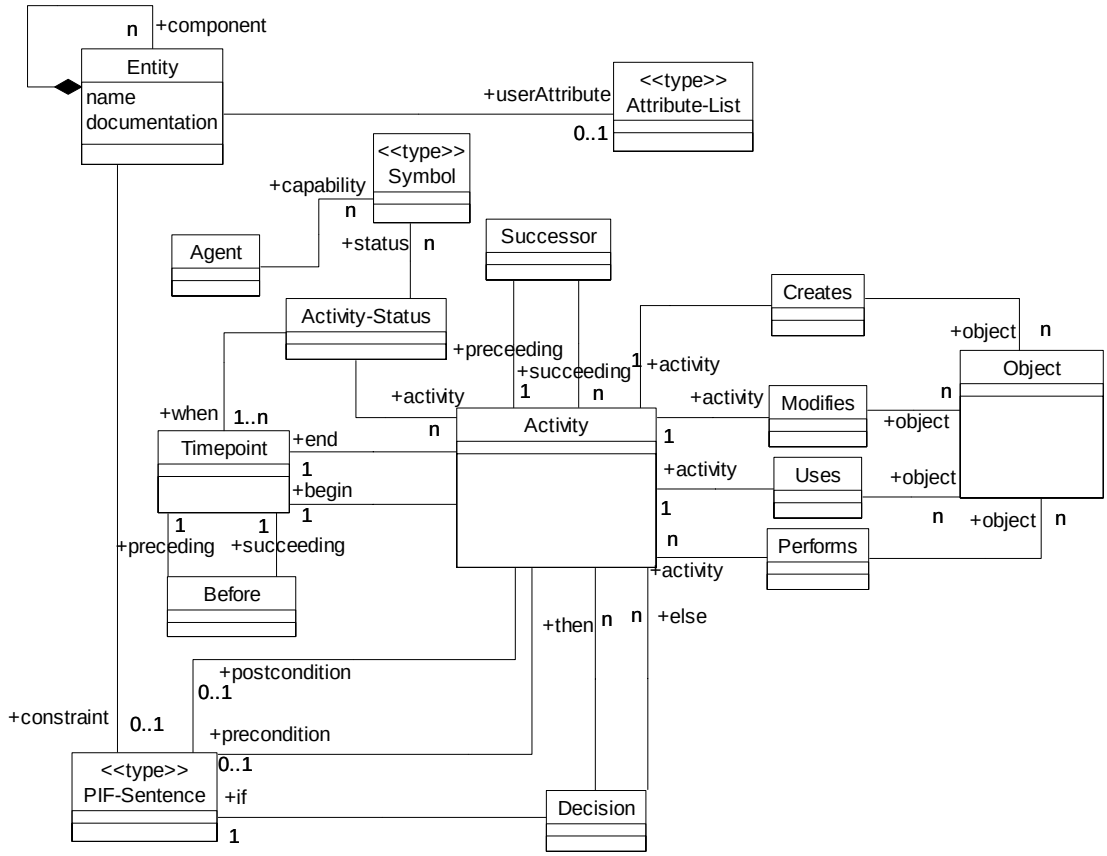


Figure 51. Relations entre entités dans PIF

PIF, tout comme PSL que nous présentons ci-dessous, propose une architecture rationalisée de méta-modèles de processus. Un noyau définit un certain nombre de concepts de base. Ce noyau peut ensuite être raffiné pour prendre en compte des aspects supplémentaires. L'intérêt d'une telle organisation est qu'elle permet de partager un ensemble de mécanismes entre plusieurs méta-modèles de processus. Sur cette base, le qualiticien, le contrôleur de gestion et l'informaticien peuvent intégrer leurs spécificités. On peut alors définir explicitement les concepts communs et ceux qui sont particuliers à chacun des contextes, et ainsi spécifier le périmètre d'activité de chacun.

4.10. PSL (Process Specification Language)

L'objectif de PSL [85] est de définir une ontologie et un format standard pour l'échange de processus industriels. PSL a récemment été adopté par l'ISO sous le nom ISO 18629 [38]. Il devient donc le langage de spécification standard des processus industriels de production.

PSL définit un méta-modèle noyau basé sur des théories fondamentales formalisant les concepts et les relations de base. En plus de ce module de base, un certain nombre

d'extensions ont été prédéfinies (Figure 52). Chacun de ces modules spécialise une des entités basiques (ainsi l'extension *ProcessorAction* spécialise le concept d'*Activity* tandis que l'extension *ResourcePools* raffine la notion d'*Object*).

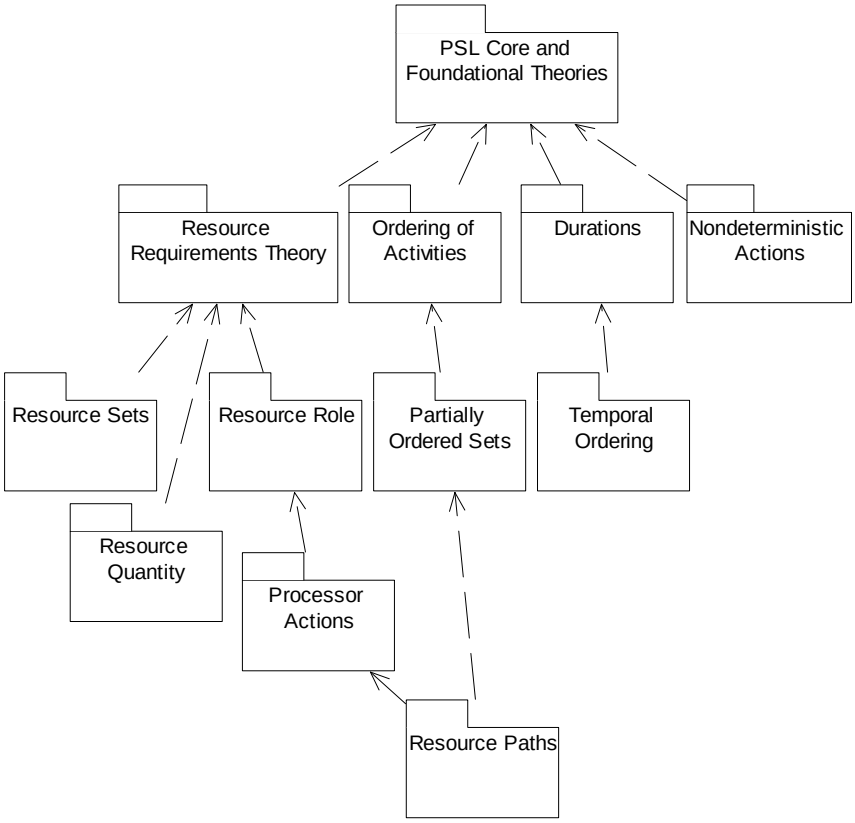


Figure 52. Architecture modulaire de PSL

L'ontologie PSL définit un processus comme un ensemble d'activités (*Activity*) dans lequelles sont impliquées des objets (*Object*) à des instants donnés (*Timepoint*) (voir Figure 53). Dans PSL tout est activité, objet ou instant. PSL introduit également la notion d'occurrence d'activité (*Activity_Occurrence*). Cette base minimale est enrichie dans des modules d'extension qui définissent par exemple des activités non déterministes, des quantités sur les objets ou des relations d'ordonnancement entre instants.

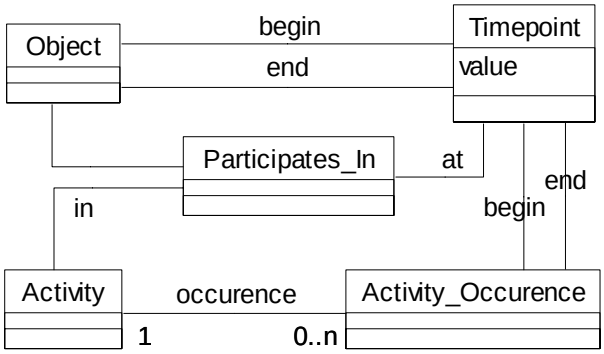


Figure 53. Noyau du méta-modèle de PSL

PSL est défini de manière précise et non ambiguë par des axiomes et des définitions exprimées en utilisant KIF. Certaines de ces spécifications peuvent être directement exprimées dans un méta-modèle. C'est par exemple le cas de l'axiome indiquant qu'une occurrence est associée avec une et une seule activité. Par contre d'autres exigent des mécanismes d'expression plus puissants tels OCL. Ainsi, l'axiome indiquant que le moment auquel débute l'occurrence d'une activité doit être antérieur ou égal au moment auquel elle termine peut être sépcifié en OCL de la façon suivante :

```
context Activity_Occurence
        inv: self.begin.value <= self.end.value
```

KIF permet donc de disposer d'un seul et unique langage pour définir à la fois la partie terminologique et la partie assertionnelle d'un formalisme. Ces deux parties sont alors présentées sous une forme homogène. Ce n'est pas le cas avec les techniques de méta-modélisation. Le langage défini par le méta-modèle ne définit que la syntaxe abstraite du méta-modèle. Il faut lui adjoindre des langages complémentaires pour prendre en compte d'autres aspects tels que l'aspect assertionnel.

4.11. SPEM (Software Process Engineering Metamodel)

SPEM est une proposition de l'OMG en cours de finalisation. L'objectif de SPEM est la définition d'un formalisme dédié à la description du processus de développement logiciel. SPEM est à la fois un méta-modèle MOF et un profil UML, des correspondances ayant été établies entre ces deux formalismes.

SPEM est découpé en sept paquetages (Figure 54). Le paquetage Data-Types introduit les types de base (*Integer*, *String*, *Boolean*, etc.). Le paquetage *Core* intègre de nombreux concepts communs avec le paquetage *Core* d'UML 1.4.. Le paquetage *Model_Management* définit la notion de paquetage. Les quatre derniers paquetages définissent les entités véritablement spécifiques.

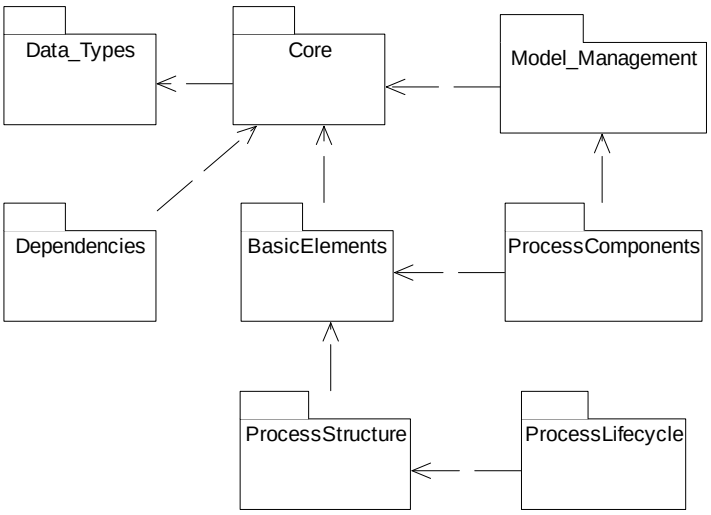


Figure 54. La décomposition de SPEM en plusieurs paquetages

Un processus (*Process*) peut être considéré comme une suite d'activités (*Activity*), réalisées par des rôles (*ProcessRole*), qui vont utiliser et consommer des produits (*WorkProduct*) et des informations (*InformationElement*). Ces produits et informations constituent les paramètres de l'activité. Une des particularités de SPEM est que l'activité est contenue par le rôle (ou le *ProcessPerformer* si aucun rôle précis n'est identifié) qui la réalise. Un rôle définit donc un ensemble d'activités, tout comme une classe UML définit des méthodes.

Les activités peuvent être regroupées selon des disciplines thématiques (*Discipline*). Elles sont en fait des définitions de travail (*WorkDefinition*), tout comme les itérations (*Iteration*), phases (*Phase*) et cycle de vie (*Lifecycle*). La seule indication sur la façon dont ces concepts s'organisent est la contrainte restreignant le cycle de vie à ne contenir que des phases. Une définition de travail a des pré-conditions (*Precondition*) et des objectifs (*Goal*), qui peuvent être vus comme des post-conditions. Ces deux éléments sont des contraintes portant sur les statuts des produits paramètres de la définition ou de l'une de ses sous-définitions.

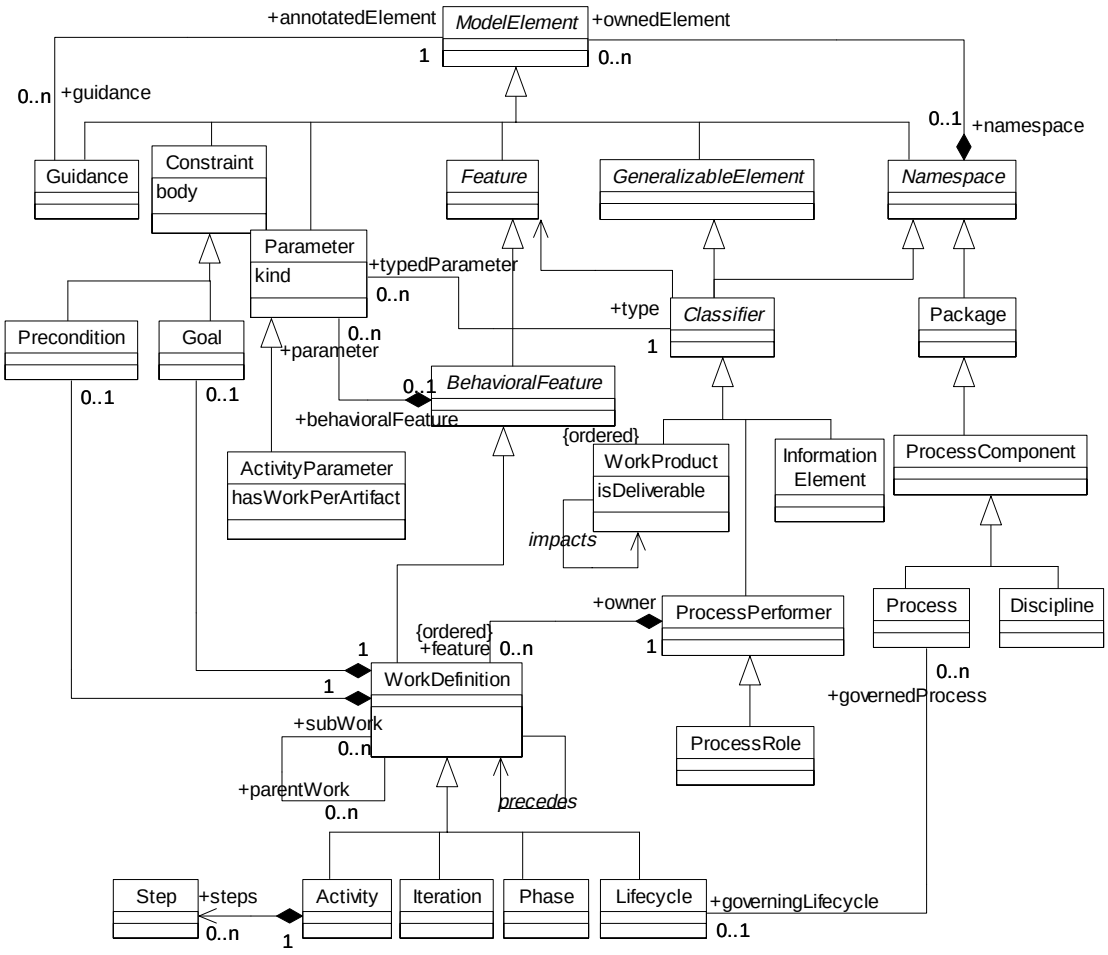


Figure 55. Le contenu de SPEM (vue partielle)

Il est possible de définir des processus de façon modulaire. Les composants de processus (*ProcessComponent*) permettent de définir des parties de processus qui pourront être intégrées ultérieurement pour constituer des processus. Ainsi, des entreprises ayant développé un savoir-faire dans une partie du processus (la spécification des besoins, les tests, etc.) pourront le distribuer sous la forme de modèles intégrables au sein du processus de développement logiciel de leurs clients. Il reste toutefois une limitation importante à ce type d'utilisation, l'unification des éléments entre le processus et les composants qu'il intègre. Pour cela, la spécification ne propose actuellement qu'un renommage manuel des éléments à unifier. Nous verrons dans la troisième partie que la réutilisabilité de parties de processus est un problème récurrent et complexe, impliquant des mécanismes de mise en correspondance.

4.12. TOVE (Toronto Virtual Enterprise)

TOVE est une ontologie définie par l'EIL (Enterprise Integration Laboratory), laboratoire qui fait partie de l'université de Toronto. TOVE est destiné à la modélisation d'entreprise. Le formalisme est découpé en un certain nombre d'ontologies interdépendantes adressant chacune un domaine bien défini tel que :

- les activités **[32]**,
- les ressources **[26]**,
- l'organisation **[28]**,
- la qualité **[44]**,
- les coûts **[95]**.

Nous décrivons ici plus particulièrement l'ontologie d'activité, qui constitue une des ontologies de base de TOVE.

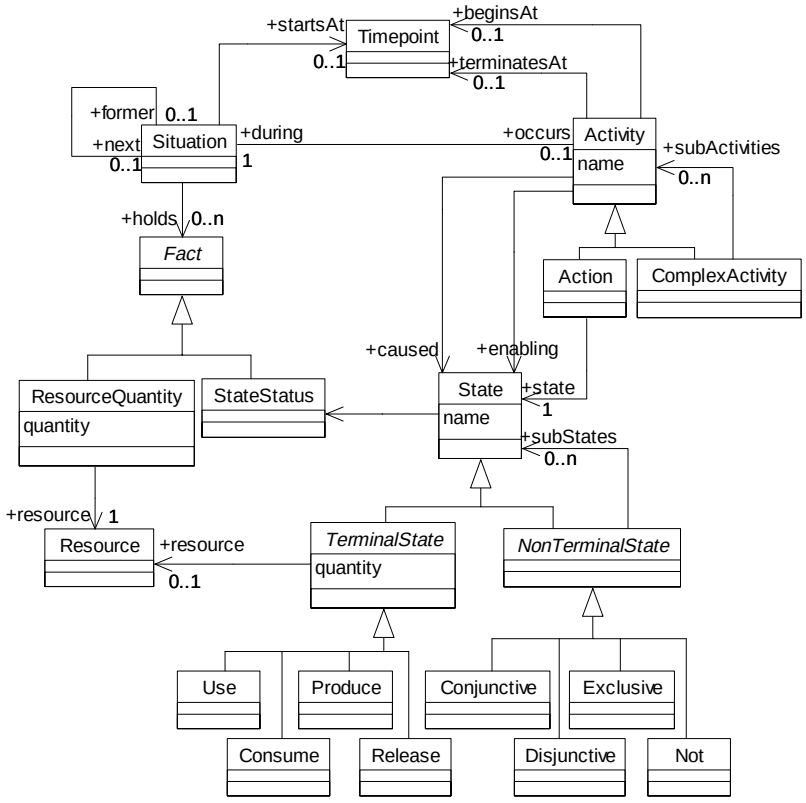


Figure 56. La définition des activités dans TOVE

Le concept de base de l'ontologie d'activité est la situation. Une situation est un ensemble de faits avérés sur une certaine période de temps. Un fait peut être la quantité de ressources disponibles ou le statut d'un état. Les activités provoquent le passage d'une situation à une autre, en introduisant des modifications dans la situation d'origine.

Les activités peuvent être définies avec plusieurs niveaux de granularité. Les activités complexes contiennent d'autres activités. Les actions constituent le niveau atomique. Une

action modifie le statut d'un état. Les activités passent par un certain nombre d'états (Figure 57). Ces états sont calculés en fonctions des statuts des états (*State*) dont ils dépendent.

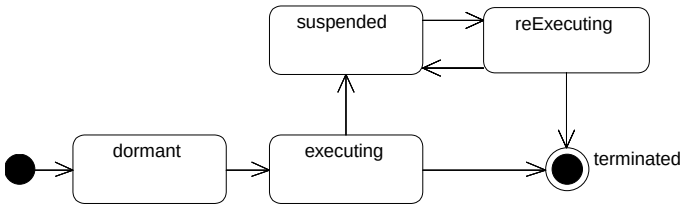


Figure 57. Le diagramme d'état des activités dans TOVE

La notion d'activité est donc fortement corrélée à la notion d'état. Au plus bas niveau une activité définit une liste de changement d'états (ce sont les actions). De plus les pré- et post-conditions des activités sont également définies par des états. Les états sont de deux types. Il y a des états complexes, qui permettent de combiner plusieurs sous-états avec les opérateurs AND (*Conjunctive*), OR (*Disjunctive*), XOR (*Exclusive*) et NOT (*Not*). Enfin, il y a les états terminaux. Deux de ces états peuvent être utilisés dans les pré-conditions, *Use* et *Consume*. Ils déterminent tous les deux des besoins de l'activité en ressource. La différence entre eux tient au fait que les ressources consommées ne peuvent être réutilisées par les activités suivantes. Les deux derniers états, *Produce* et *Release*, sont utilisés dans les post-conditions. Ils définissent les quantités de ressources qui sont produites ou libérées à la fin de l'activité. Les états prennent un certain nombre de statuts (Figure 58). Le statut d'un état complexe est calculé à partir des statuts de ces sous-états.

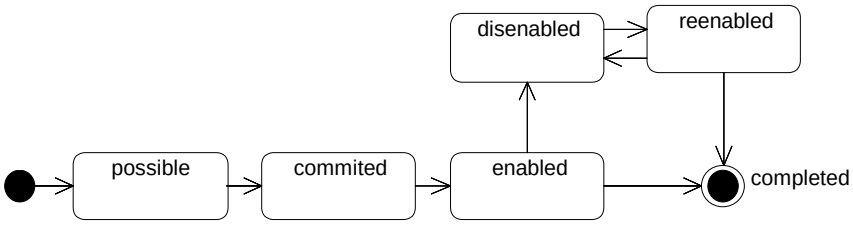


Figure 58. Le diagramme d'état des états dans TOVE

Il n'y a pas de notion de précédence entre les activités. Les seules informations indiquant quand une activité doit démarrer sont spécifiées dans les pré-conditions. Il est toutefois possible dans certain cas de mettre cette relation en évidence, par exemple quand une activité A a pour post-condition l'état pré-condition d'une activité B.

4.13. UML (Unified Modeling Language)

UML est avant tout dédié à la représentation des artefacts des logiciels à objets. Le diagramme d'activité était initialement prévu pour la représentation des processus de traitement internes aux applications. Toutefois, du fait des capacités d'extension d'UML, et de la généricité des concepts qu'il définit, de nombreuses propositions ont été faites pour l'utiliser pour la modélisation de processus. On peut citer entre autre [23] et [36]. De nombreux outils du marché se servent également des diagrammes d'activités d'UML pour

définir des processus (Business Designer de WebGain). Enfin, on a vu que de nombreux profils sur le sujet étaient en cours de définition.

Le graphe d'activité UML est en fait une extension du diagramme d'état. Dans la figure suivante (Figure 59), les entités qui ne sont pas directement comprises dans le paquetage *Activity Graphs* ont été grisées.

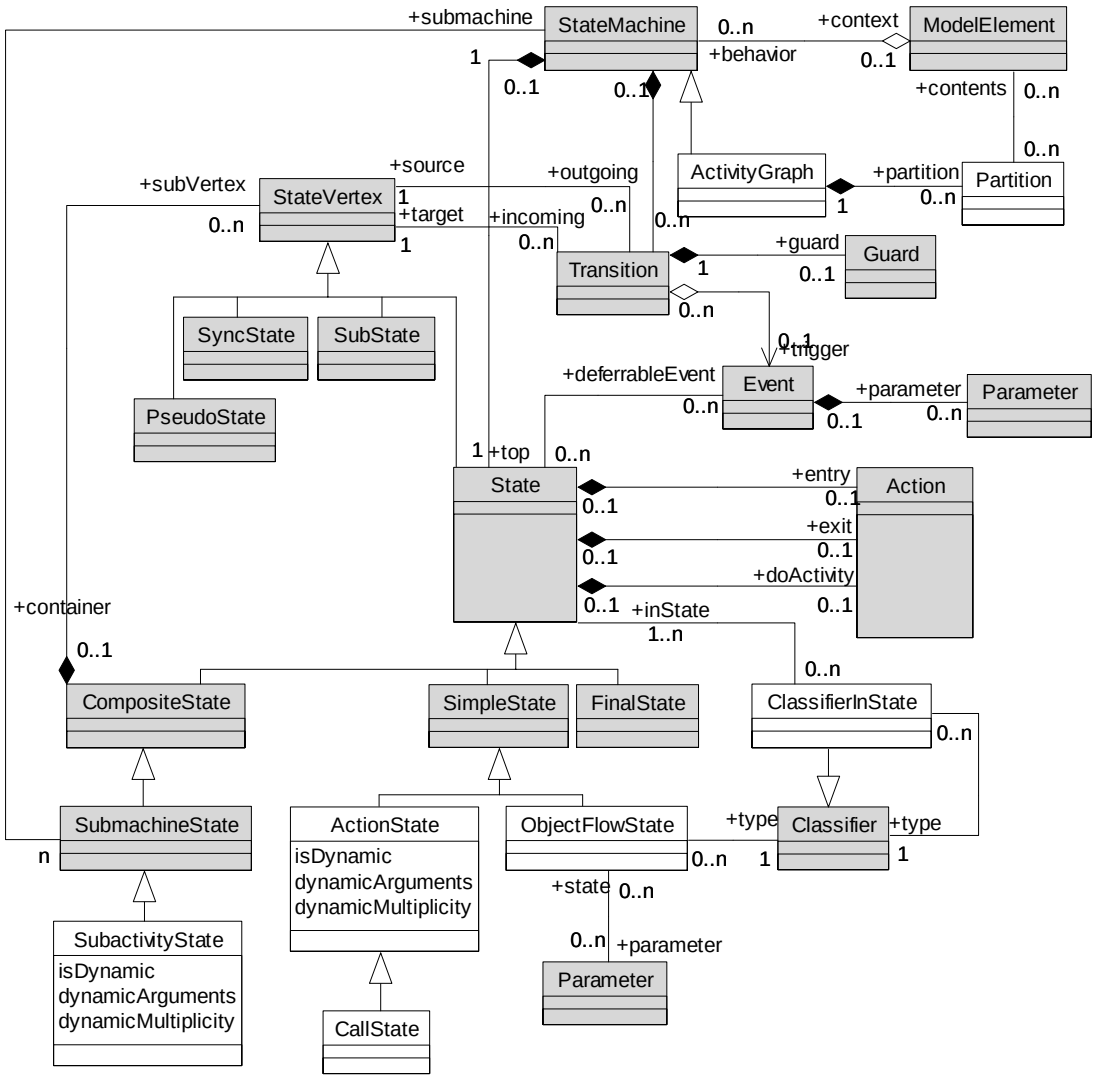


Figure 59. Définition du graphe d'activités dans UML

Un graphe d'activité (*ActivityGraph*) est un sous-type de la machine à états (*StateMachine*). Il peut être rattaché à un paquetage, un *Classifier* (comme une classe ou un cas d'utilisation), ou une *BehavioralFeature* (comme une méthode). Il permet de décrire un processus de traitement informatique. Il peut également être utilisé pour représenter des processus d'entreprise.

Il définit donc des états (*State*) tels que les *PseudoState initial*, *join* ou *fork* ou encore l'état de fin (*FinalState*). Un certain nombre d'états supplémentaires ont été introduits au

niveau du paquetage *Activity Graphs*. L'*ActionState* est une activité atomique correspondant à l'exécution de l'action (*Action*) reliée par le lien *entry*. Un *CallState* est une activité atomique particulière reliée à une *CallAction*, et qui invoque donc une opération sur une instance. Le *SubactivityState* définit un lien vers un sous-graphe d'activités. *ActionState* et *CompositeState* peuvent être éventuellement exécutés plusieurs fois en parallèle (propriété *isDynamic*). Ce nombre de fois peut être limité (propriété *dynamicMultiplicity*). Enfin, des arguments propres à chaque exécution peuvent être spécifiés (propriété *dynamicArguments*).

Un graphe d'activité peut être divisé en un certain nombre de partitions (*Partition*). Celles-ci permettent de regrouper les activités de façon logique. Elles peuvent correspondre à indiquer des allocations de ressources communes, des affectations au sein d'unités organisationnelles ou encore à marquer le fait que certaines activités partagent des caractéristiques communes. Bien que la partition serve le plus souvent pour désigner le rôle responsable de la réalisation de l'activité, cette utilisation n'est pas la seule possible.

La définition du graphe des activités peut se faire comme pour les états, au moyen de transitions (*Transition*) et d'événements (*Event*). La fin d'une activité, l'occurrence d'un signal ou la satisfaction d'une certaine condition peuvent amener à examiner des transitions, et à les activer dans le cas où leur garde (*Guard*) serait avérée. Dans le cas d'une action atomique, seul l'événement implicite de complétion déclenche les transitions en sortie.

En plus de la définition du flux de contrôle, on peut également définir un flux de données entre activités. L'entité *ObjectFlowState* définit un flux d'objet. Cet objet est d'un type prédéfini (*type*), voire même dans un certain état. L'entité *ClassifierInState* spécifie à la fois le type de l'objet et l'état dans lequel celui-ci doit se trouver. *ObjectFlowState* sert donc tout à la fois pour définir la production ou la modification d'un objet par une activité, son utilisation dans une autre activité, et le transfert de l'objet entre les activités concernées. Il peut également être utilisé pour définir des synchronisations entre activités de type producteur/consommateur (propriété *isSync*).

4.14. WPDL (Workflow Process Definition Language)

Le Workflow Management Coalition (WfMC) est un consortium international d'éditeurs de worfklow, d'utilisateurs, d'analystes et de groupes de recherches dont les objectifs sont la promotion des technologies de workflow et la définition de standards. En particulier, le WfMC propose un méta-modèle de définition de processus [106] associé à un langage, le WPDL. Ce dernier se positionne comme un format d'échange entre les outils de modélisation et les moteurs d'exécution. Dans la pratique, chaque produit commercialisé propose son propre formalisme. La méta-modélisation a d'ailleurs déjà été utilisée comme un outil permettant la comparaison entre ces formalismes (voir [47], [108]). Le méta-modèle de processus de la WfMC (voir Figure 60) est totalement défini dans un unique module.

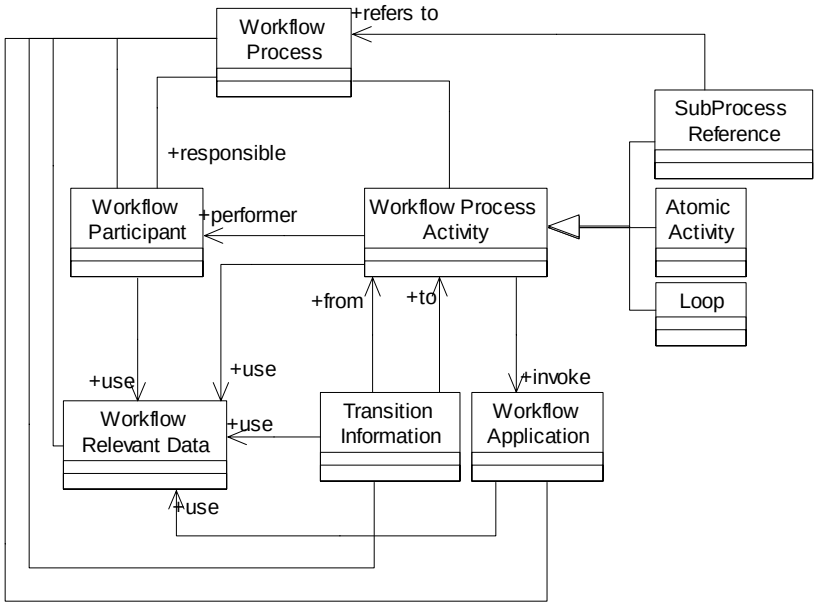


Figure 60. Noyau du méta-modèle de processus de la WfMC

Le *Workflow Process* représente le processus complet. Il est divisé en plusieurs activités (*Workflow Process Activity*) qui peuvent être atomiques (tâches élémentaires) ou être des appels vers des sous-processus. Ces activités sont ordonnancées grâce aux informations de transition (*Transition Information*) qui peuvent porter sur les données pertinentes pour le workflow (*Workflow Relevant Data*). Ces données sont typées, un ensemble de types de base ayant été défini (chaîne de caractères, entier, réel, date). Les activités sont réalisées par des ressources humaines ou des entités organisationnelles définies par l'intermédiaire des participants (*Workflow Participant Specification*). Elles peuvent invoquer des applications externes via des déclarations d'application (*Workflow Application Declaration*). Le diagramme d'état définissant l'exécution des activités est présenté Figure 61. Il est intéressant de noter que le concept de donnée pertinente pour le workflow ne recouvre pas l'ensemble des données manipulées pendant un processus, mais la sous-partie de ces données qui est utilisée dans les conditions de transition, dans les règles d'affectation d'une tâche à un participant ou dans l'invocation d'une application externe.

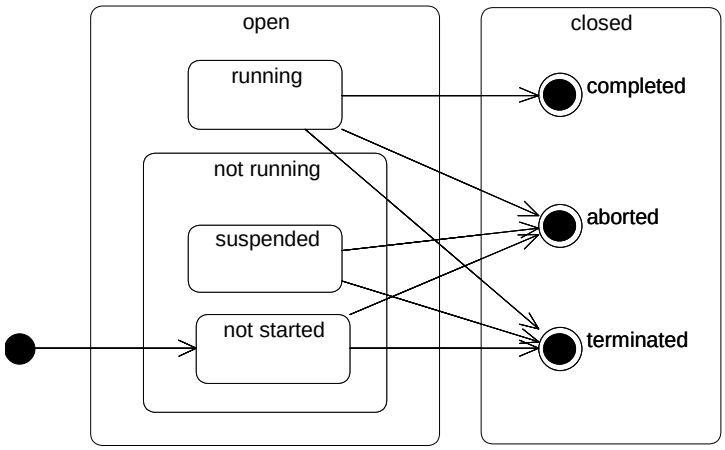


Figure 61. Diagramme d'état des activités dans WPDL

Ce méta-modèle de définition de workflow redéfinit donc des mécanismes de typage des données et de définition de structure organisationnelle. Ces mécanismes sont simplifiés à l'extrême. Pour pallier le manque de richesse de la définition de la structure organisationnelle, la spécification prévoit la possibilité d'une mise en relation avec un modèle organisationnel. Cela permettrait de formaliser des règles bien plus complexes que ce que permet le modèle de base. On a donc l'exemple dans ce méta-modèle de zones de recouvrement avec d'autres domaines (produit et structure organisationnelle). L'introduction de ces concepts, même de façon sommaire, permet de décrire des processus exécutables à partir de ce seul et unique méta-modèle. Toutefois, une architecture rationalisée de méta-modélisation plaiderait pour une définition explicite des relations entre ces différents méta-modèles.

4.15. WSFL (Web Service Flow Language)

WSFL [50] est un dialecte XML développé par IBM et destiné à la composition de services Web. Deux types de composition ont été identifiés : les processus et les modèles d'interaction. Dans notre étude de WSFL, nous nous intéresserons principalement aux processus. WSFL est une extension de WSDL [101] (Web Service Definition Language), qui est une recommandation du W3C pour la description des services Web.

WSDL définit un service Web comme un ensemble d'opérations (*Operation*) regroupées par port (*Port*), un service (*Service*) étant un ensemble de ports (voir Figure 62). Une opération accepte un message (*Message*) en entrée (*Input*), et produit un message en sortie (*Output*). Elle peut également produire des messages d'erreur (*Fault*). Les opérations sont en fait regroupées par type de port (*PortType*). Les ports sont reliés aux types de port par des *Binding* qui permettent de préciser les protocoles et les formats de message utilisés pour chacune des opérations du type de port. Le type de port représente l'interface du service, alors que le port définit une implémentation particulière.

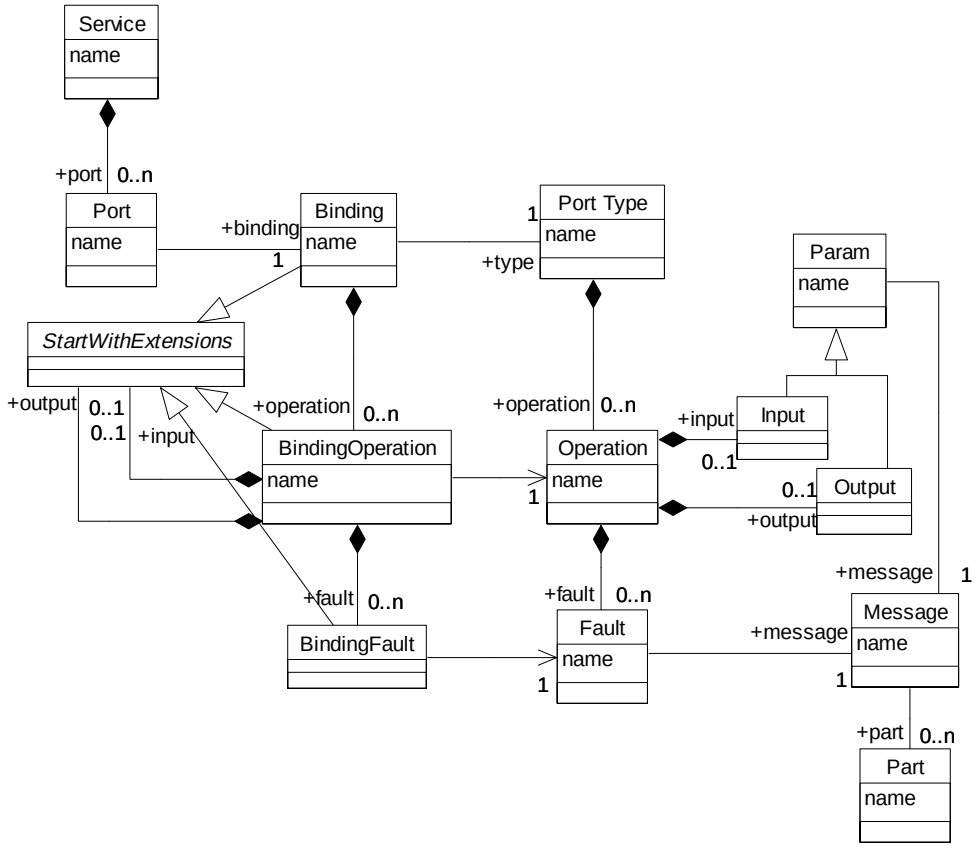


Figure 62. Le méta-modèle de WSDL

Un des objectifs de WSFL est donc d'organiser des services Web en processus. La description des processus se fait à l'aide des modèles de flux (*FlowModel*). Un modèle de flux est un ensemble d'activités (*Activity*), de liens (*Link*) et de fournisseurs de services (*ServiceProvider*). Enfin, un modèle de flux a une interface en entrée (*FlowSource*) et une en sortie (*FlowSink*). Les fournisseurs de services capturent la notion de rôle. Les activités décrivent les tâches devant être réalisées. Une activité est implémentée par une opération WSDL. Cette opération peut soit être définie de façon interne (*Internal*), soit être exportée d'un fournisseur externe (*Export*). Ce sont des opérations WSDL (*Operation*). Une activité peut donc avoir un message en entrée, un message en sortie, et des messages d'erreur. Cette signature est bien sûr reliée à la signature de l'opération qui l'implémente. Le moment où une activité devient active est définie grâce aux liens de contrôle (*ControlLink*). Ces liens peuvent définir une condition de transition. Une activité peut être la cible de plusieurs liens de contrôle. Elle peut les synchroniser ou non, selon ce qui est défini dans la jointure (*Join*). Enfin, une activité va recevoir des messages issus d'autres activités et en produire en direction d'activités ultérieures. Ce flux d'information est défini par les *DataLink*, qui spécifient également la transformation à effectuer pour extraire les informations du message source et alimenter le message cible (*Map*). De tels flux ne peuvent être définis qu'entre activités reliées, directement ou indirectement, par des liens de contrôle. Comme une activité peut être réceptrice de plusieurs flux de données, il est nécessaire d'indiquer la façon dont ils

sont intégrés. Cela peut se faire selon un ordre défini (champ *order* de *MapPolicy*), ou bien en utilisant des fonctionnalités externes comme un fichier XSLT (*Construction*).

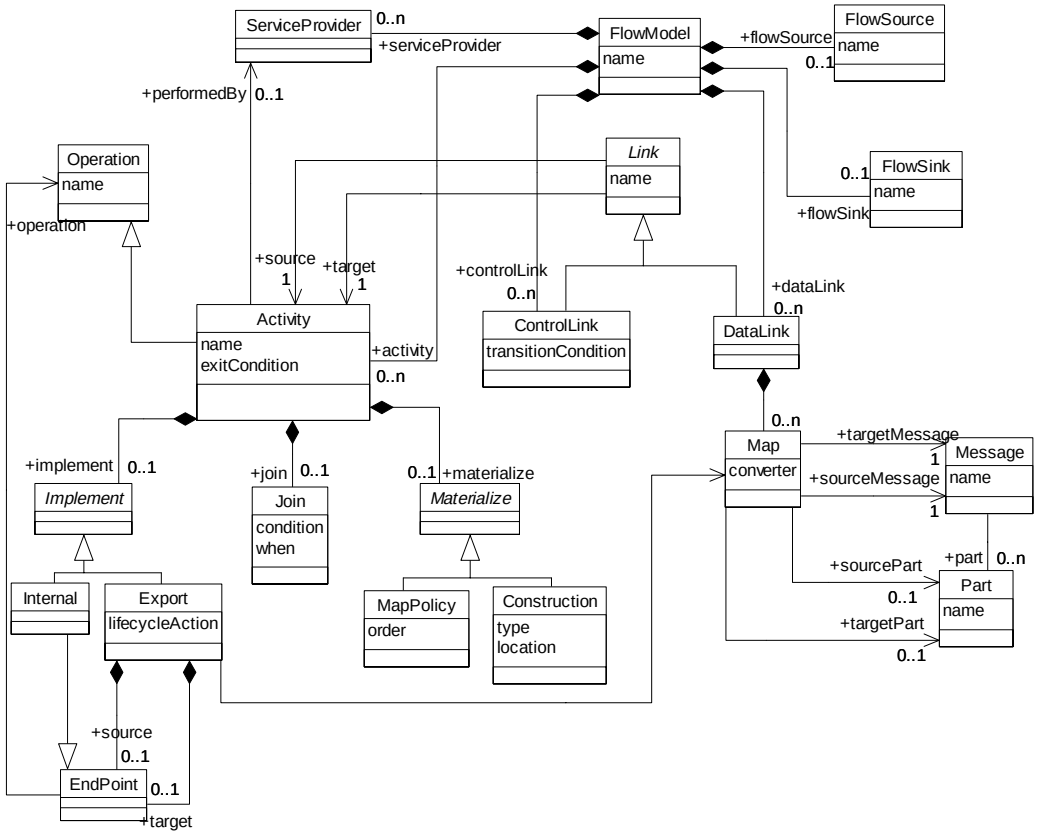


Figure 63. Le méta-modèle de WSFL

Les cycles de vies suivants ont été définis pour les processus (Figure 64) et pour les activités (Figure 65).

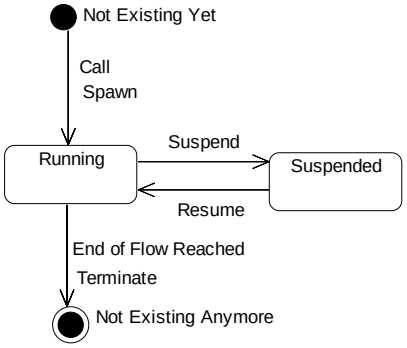


Figure 64. Diagramme d'état d'un processus (*FlowModel*)

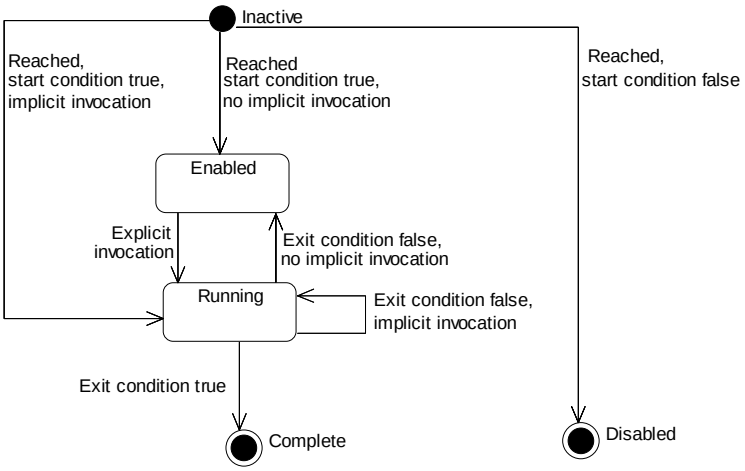


Figure 65. Diagramme d'état d'une activité

4.16. XLANG

XLANG [96] est un dialecte XML développé par Microsoft, dont l'objectif est de permettre la description de processus se basant sur des services Web. C'est le langage de définition de processus utilisé au sein de la plate-forme BizTalk. XLANG est une proposition directement concurrente de WSFL. Tout comme WSFL, XLANG est une extension de WSDL. On peut d'ailleurs s'attendre à ce que ces deux propositions fusionnent rapidement.

Un service XLANG est un service WSDL qui spécifie un comportement (*Behavior*) (Figure 66). Un comportement est constitué d'un processus et d'un ensemble de corrélations. Il y a huit types de processus identifiés :

- *All* : indique une exécution parallèle des sous-processus
- *Compensate* : invoque la compensation d'une transaction
- *Context* : un contexte forme un cadre pour des déclarations locales, la gestion des transactions et la gestion des exceptions.
- *Empty* : un processus vide
- *Pick* : attend l'arrivée d'événements (*Event*) pour déclencher des processus (un par événement) ; cet événement peut être la levée d'un signal ou la fin d'une action
- *Sequence* : une séquence de processus et d'actions
- *Switch* : définit des branches conditionnelles, chacune reliée à un processus
- *While* : définit une boucle sur un processus

Il y a quatre types d'action définis en XLANG :

- *Action* : invoque une opération d'un service Web

- *DelayUntil* et *DelayFor* : suspend l'exécution du processus
- *Raise* : lève un signal

Enfin, les corrélations déterminent les jetons qui serviront à identifier l'instance de processus en cours (par exemple le numéro d'une demande d'achat) et qui transiteront d'actions en actions.

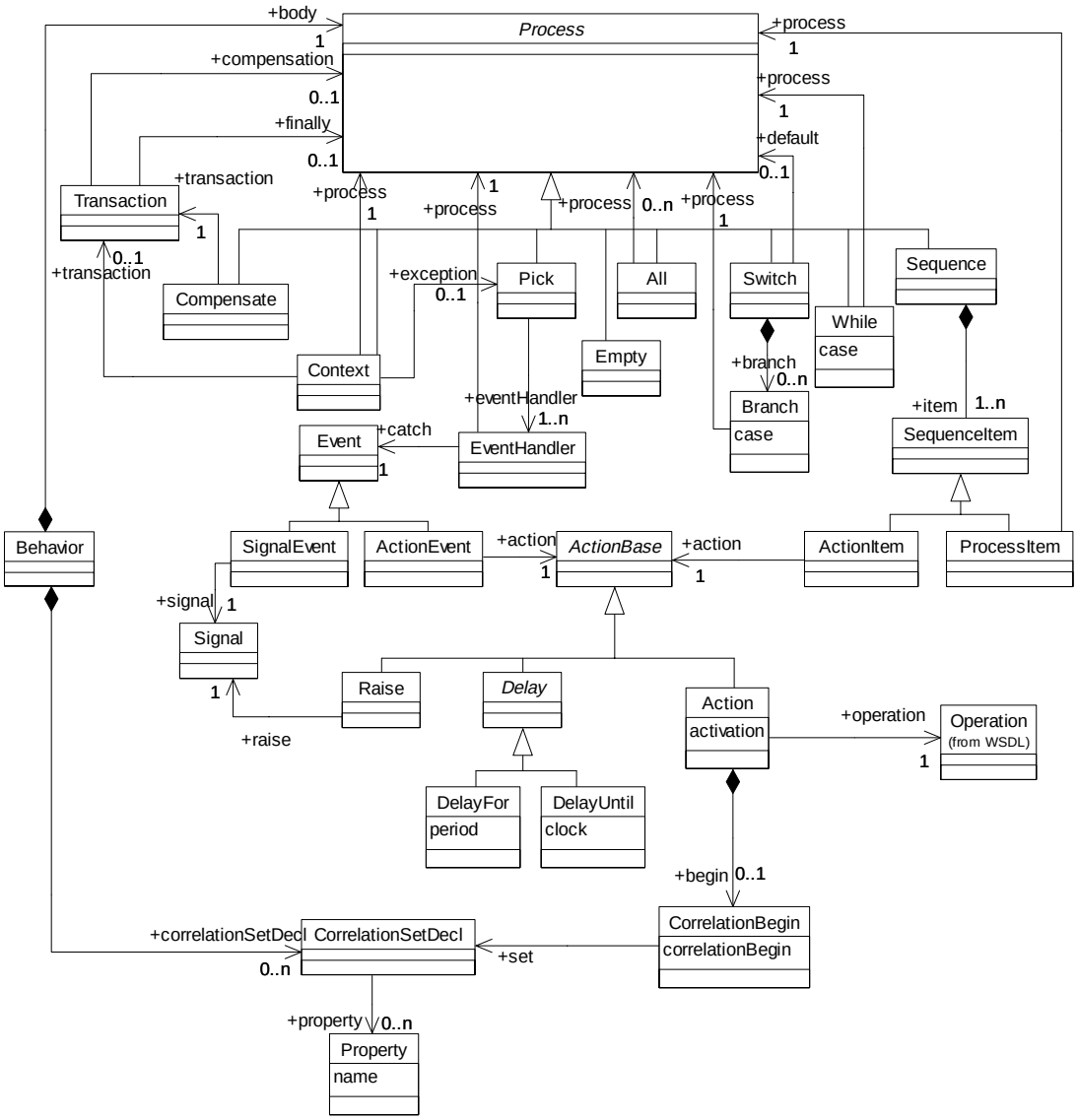


Figure 66. Le méta-modèle de XLANG

Il est intéressant de noter que le modèle d'exécution de XLANG est basé sur le Pi-Calcul, extension du modèle CCS développé par Robin Milner (voir le chapitre 5). C'est d'ailleurs également le cas de BPML. On a donc ainsi deux méta-modèles différents, chacun introduisant des concepts spécifiques, mais basés sur une même algèbre de processus. Nous n'avons toutefois pas trouvé dans la littérature la manière dont été réalisée la

correspondance entre le modèle de définition (le méta-modèle) et le modèle d'exécution (le Pi-Calcul).

4.17. Conclusion

Dans cette section, nous avons vu un certain nombre de formalismes introduisant les concepts de processus et d'activité. Ces formalismes ont été définis en utilisant des techniques bien différentes. On peut citer entre autres :

- Des dialectes XML basés sur des DTD ou sur des schémas (BPML, ebXML, WSFL, XLANG)
- Des ontologies définies en KIF (PIF, PSL)
- Des langages formels (WPDL)
- Des méta-modèles basés sur le MOF (EDOC, SPEM, UML) ou sur des formalismes équivalents (ARIS)
- Des profils UML (EDOC, SPEM)

Toutefois, dans la plupart des documents présentant ces spécifications, plusieurs formalismes différents sont utilisés pour mieux communiquer l'information. Ainsi, ebXML est-il présenté sous la forme d'un modèle UML. WPDL et PIF ont également recours à un formalisme graphique de type MOF ou UML afin de mettre en évidence les concepts qui les composent et leur relations. Enfin les spécifications de l'OMG portant sur les profils UML présentent les formalismes sous la forme de méta-modèles à part entière, au même titre qu'UML. La correspondance entre entités du méta-modèle et du profil n'est spécifiée qu'ensuite.

Un intérêt des langages de modélisation graphique est donc leur lisibilité. Le formalisme défini peut être communiqué tel quel. Au contraire, les langages textuels se prêtent mal à la communication. Lorsqu'on cherche à définir un formalisme à l'aide d'un langage de ce type, il est souvent nécessaire d'avoir recours à un langage graphique pour faciliter la compréhension.

5. Les bases de l'exécution de modèles

5.1. Introduction

La définition d'un modèle de processus peut se faire à l'aide d'un des formalismes parmi ceux que nous avons étudiés précédemment. Toutefois, le formalisme à lui seul ne suffit pas. Il faut également disposer des règles permettant de l'interpréter. Cela est bien sûr vrai lors de la définition du modèle. Il est difficile de décrire un processus si l'on ignore la manière dont un processus et une activité sont effectivement exécutés. Les lecteurs du modèle doivent être en mesure de l'interpréter de la même façon. A un autre niveau, les responsables de la définition du méta-modèle doivent pouvoir confronter leurs différentes visions et parvenir à un consensus tant sur un ensemble de concepts que sur leur utilisation. Enfin, en plus des agents humains, les modèles de processus peuvent également être destinés à des composants logiciels (moteur d'exécution, de simulation, etc.). Pour que ces derniers puissent être en mesure d'interpréter le modèle, il faut donc que les règles aient été implantées au préalable dans le composant ou qu'il soit en mesure de les intégrer dynamiquement. Pour toutes ces raisons, une spécification précise de la sémantique d'exécution des méta-modèles de processus est nécessaire. C'est d'ailleurs ces mêmes raisons qui ont poussé aux nombreux travaux sur la sémantique des langages de programmation.

Dans ce chapitre nous nous intéressons donc à l'exécution des modèles de processus. Nous commençons par présenter la vision pragmatique de l'automatisation des processus, telle qu'elle est répandue dans le domaine du workflow ou de l'EAI. Nous introduisons ensuite brièvement des concepts que nous jugeons proches : l'exécution de programme et l'opérationnalisation de connaissances. Nous étudions par la suite, à titre d'exemple, des travaux formels sur la représentation et l'exécution de processus basés sur des algèbres. Enfin, nous terminons ce chapitre en présentant des travaux sur la sémantique des actions qui ont alimenté nos réflexions en la matière.

5.2. Exécution de modèles

5.2.1. Automatisation de processus

L'automatisation d'un processus consiste à gérer de façon automatisée l'exécution d'un processus. Le composant chargé de ce contrôle, qu'il s'appelle moteur de workflow ou

système de gestion de processus, interprète les spécifications du modèle en réaction à l'occurrence d'événements externes (un processus est démarré, une tâche est terminée) ou internes (un timer est arrivé à échéance), et en fonction d'un contexte particulier. Le moteur de processus fait donc le lien entre le modèle et le monde réel (utilisateur et système d'information), comme le montre la Figure 67 représentant le modèle de référence du WfMC [35].

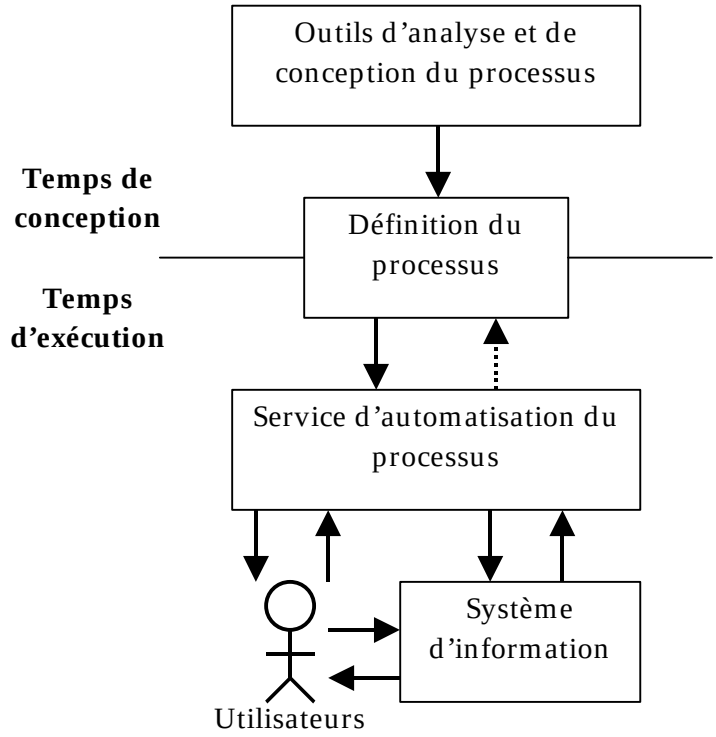


Figure 67. Le modèle de référence du WfMC

Un moteur de processus va donc proposer un certain nombre d'interfaces. Le WfMC en a identifié cinq différentes :

- L'interface 1 définit un méta-modèle, un langage et un ensemble d'APIs pour l'échange de modèles de workflow
- L'interface 2 définit une API pour accéder aux fonctions du moteur de workflow (démarrer un processus, terminer une activité, etc.)
- L'interface 3 définit les communications avec les applications invoquées
- L'interface 4 est une spécification pour l'interopérabilité entre moteurs de workflow
- L'interface 5 définit les événements historisés au cours de l'exécution d'un processus

Ces spécifications sont plus ou moins respectées par l'ensemble des systèmes de gestion de processus industriels, qui en implémentent chacun un spectre plus ou moins étendu.

Dans la plupart des environnements de gestion de processus, il y a une distinction bien nette entre la phase de modélisation et celle d'exécution. Un modèle complet et valide est un

pré-requis à l'exécution. Une exécution ne concerne qu'un modèle unique et figé. Toute modification de ce modèle aboutit donc à un nouveau modèle qui n'impacte pas les exécutions déjà en cours. Il existe des environnements de gestion de processus dits flexibles qui n'intègrent pas de telles contraintes. Le modèle peut être modifié à tout moment, mais ce n'est pas sans poser un certain nombre de problèmes de maintien de la cohérence. Les moteurs de processus compilant les modèles de processus, c'est-à-dire transformant la représentation externe en une représentation interne, appartiennent plutôt à la première catégorie. Ceux qui interprètent le modèle peuvent plus facilement intégrer des éléments de flexibilité.

5.2.2. L'exécution de programmes

Il nous semble pertinent d'établir un parallèle entre l'exécution d'un processus et l'exécution d'un programme. Tout d'abord, comme nous l'avons vu dans la section sur les techniques de méta-modélisation, il existe une correspondance entre grammaires et méta-modèles. La relation entre un programme et ses exécutions est la même que celle existant entre un modèle de processus et ses exécutions. Les différentes technologies que sont la compilation et l'interprétation sont également utilisées pour l'exécution de processus. Ainsi, dans le monde du workflow, Dimitrios Georgakopoulos et al. distinguent deux types de systèmes d'exécution de workflow, ceux qui couplent faiblement les spécifications et l'implémentation, et ceux qui les couplent fortement [29]. Dans le premier cas, l'implémentation est faite par des informaticiens dans un langage de programmation quelconque. Dans le second cas, le modèle est directement exploité. Il peut être utilisé pour générer du code exécutable. Il est en quelque sorte compilé dans un langage machine de plus bas niveau que peut exécuter le moteur de workflow, qui joue alors le rôle de processeur. Il peut également être directement interprété. Le formalisme ayant servi à le définir est donc directement exécutable, le moteur de workflow pouvant alors être comparé à une machine virtuelle.

5.2.3. L'opérationnalisation de connaissances

L'opérationnalisation est un terme utilisé en ingénierie des connaissances. Il désigne le passage du modèle conceptuel au composant exécutable. Les modèles conceptuels peuvent être soit des modèles de domaine, soit des modèles de raisonnement. Le modèle du domaine décrit les connaissances relatives au domaine. Le modèle de raisonnement décrit les résolutions de problème. Gilles Morel et Gilles Kassel [61] ont identifié trois types de langages de spécification dans le domaine de l'ingénierie des connaissances :

- Les langages de modélisation : généralement non formels mais lisibles et compréhensibles
- Les langages de formalisation : formels, ce qui permet de supprimer toute ambiguïté, mais non exécutables
- Les langages d'opérationnalisation : associant à chaque primitive de modélisation les structures de données et les traitements nécessaires à leur mise en œuvre informatique.

Dans leur comparaison des différents langages formalisant et opérationnalisant KADS (Knowledge Analysis and Design Support), Dieter Fensel et Franck van Harmelen [27] distinguent également spécifications formelles et opérationnelles, tout en précisant que des correspondances peuvent être établies entre une spécification formelle et une spécification opérationnelle, permettant ainsi le prototypage.

5.3. Des travaux de formalisation sur les processus

Il existe de nombreux travaux de sur la formalisation des processus qui se base sur une algèbre. Dans cette section, nous présentons à titre d'exemple deux propositions importantes : celles de Tony Hoare et de Robin Milner. Nous n'évoquons pas ici les réseaux de Petri, autre proposition importante, mais nous les étudions en détail dans la troisième partie de notre document. Il existe également des travaux plus spécialisés autour de la définition d'une sémantique d'exécution des workflows. On peut par exemple citer les travaux de Munindar Singh qui a défini une algèbre pour la formalisation des dépendances entre tâches ([87], [88]).

5.3.1. Le modèle CSP

« Communicating Sequential Processes » est le titre du livre de Tony Hoare [34] dans lequel celui-ci présente une théorie mathématique pour le calcul de processus. Cette théorie trouve ses sources dans les travaux sur les langages CSP et OCCAM.

Un processus est défini comme le modèle de comportement d'un objet. Il est spécifié sous la forme d'un ensemble limité d'événements qui constitue son alphabet. La séquence *(x -> P)* décrit un processus qui débute par l'événement *x* puis se comporte comme il est spécifié par *P*. La définition d'un processus peut être récursive, c'est-à-dire qu'on peut avoir *P = (x- > P)*.

Des processus peuvent être composés pour en former de plus complexes. Hoare a défini un certain nombre d'opérateurs, chacun d'eux spécifiant des comportements différents :

- **« | »** définit un choix entre deux processus selon l'occurrence d'un événement initial. *(x -> P | y -> Q)* décrit un processus qui se comporte comme *P* ou comme *Q*, selon qu'arrive l'événement *x* ou l'événement *y*.
- *(P* ⊓ *Q)* définit un choix non déterministe entre les deux processus *P* et *Q*, indiquant que la sélection entre les deux processus est faite de façon arbitraire, sans aucune intervention de l'environnement externe.
- *(P* [] *Q)* est une généralisation du choix. Il est déterministe si aucun événement initial n'est commun entre les deux processus pouvant être sélectionnés. Il revient alors à *(x -> P | y -> Q)*. Il est non déterministe si un même événement peut indifféremment démarrer l'un ou l'autre des processus, et dans ce cas il est similaire à *(P* ⊓ *Q)*.

- *(P □ Q)* décrit une interaction entre les deux processus *P* et *Q*, nécessitant la participation simultanée des deux processus à chaque occurrence d'événements communs à leurs deux alphabets.
- *(P ; Q)* définit une exécution séquentielle de *P* et *Q*.

Un certain nombre d'autres opérateurs ont été définis pour spécifier l'interruption d'un processus par un autre, le démarrage d'un processus d'exception, etc. Sur ces descriptions de processus on va ensuite pouvoir appliquer des calculs afin de les simplifier ou d'identifier des risques de blocage.

L'échange de message entre processus est un événement particulier. Il se fait par l'intermédiaire de canaux. Un processus peut émettre ou recevoir des messages. *c!m* décrit l'émission d'un message *m* sur le canal *c*, tandis que *c?m* représente la réception d'un message *m* sur le canal *c*.

Les processus ainsi exprimés sont des modèles. Les événements qui composent leur alphabet sont en fait des classes d'événements. Une exécution réelle d'un processus peut être stockée au moyen de traces. Une trace du comportement d'un processus est la séquence finie des événements dans lesquels il s'est engagé à un moment dans le temps. Ces traces peuvent être contraintes par des spécifications, ensemble d'assertions permettant de réduire le nombre de comportements possibles du processus. On dit que le processus satisfait une spécification si l'on peut prouver que la contrainte est respectée pour l'ensemble des traces possibles du processus.

A partir des concepts que nous avons présentés précédemment, on peut établir le méta-modèle suivant (voir Figure 68). Un processus est soit un processus simple, c'est-à-dire un événement suivi d'un autre processus, soit un processus complexe (choix, parallélisme ou séquence) ou une fin de traitement. Les opérateurs de choix, de parallélisme et de séquence étant associatifs, nous avons pu spécifier qu'un processus complexe peut intégrer de 2 à n sous-processus (seule contrainte, ils doivent être ordonnés dans le cas de la séquence). Enfin, les entrées-sorties sont des types particuliers d'événements. Une entrée-sortie définit l'émission ou la réception d'un message sur un canal.

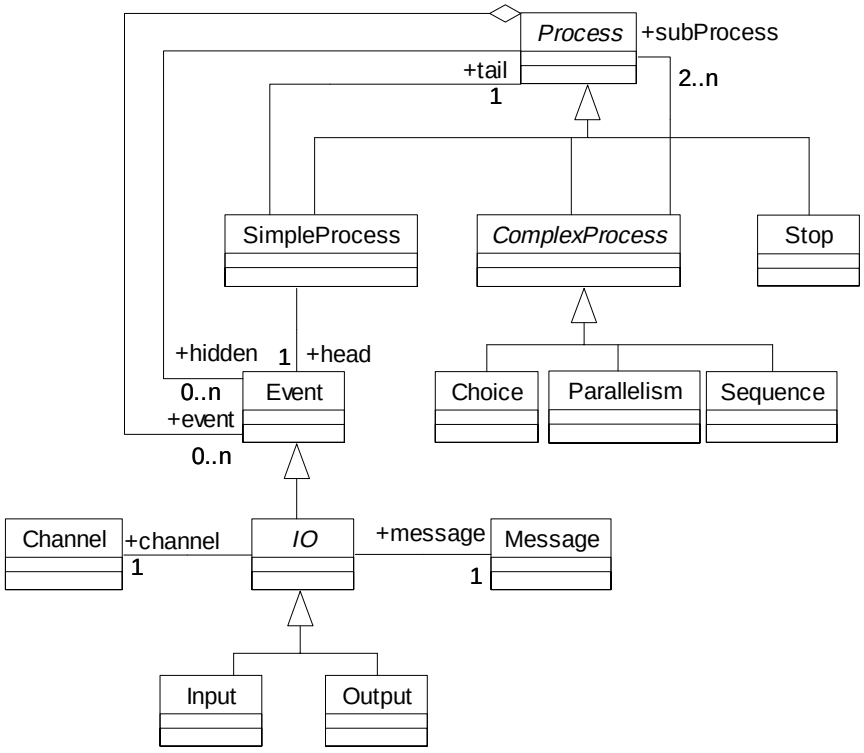


Figure 68. Méta-modèle (partiel) du modèle CSP

5.3.2. Le modèle CCS

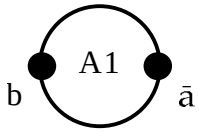
Les travaux de Robin Milner présentés dans l'ouvrage « Communication and Concurrency » [60] établissent une théorie des systèmes communicants. Le langage CCS est une implémentation de ces travaux. Une extension de CCS, le PI-Calculus, définit le modèle d'exécution de BPML et de XLANG.

Les entités de base de CCS sont les agents. Un agent recouvre un certain nombre d'actions. Son comportement est spécifié par l'assemblage en séquence de ces actions (opérateur '.'). Le modèle CCS s'intéressant plus particulièrement aux communications, deux types d'actions ont été distingués, les actions émettrices et les actions réceptrices. La synchronisation entre agents se fait par des actions complémentaires, c'est-à-dire deux actions qui portent le même nom, dont l'une est émettrice et l'autre réceptrice. Afin de les distinguer, le nom de l'action émettrice est surlignée. L'agent A1 définit la séquence d'actions $\bar{a}$, b, puis se termine, ce qui est indiqué par le symbole '0'.

A1 = $\bar{a}$. b . 0

A2 = a . c . 0

A1 peut être représenté graphiquement de la façon suivante :



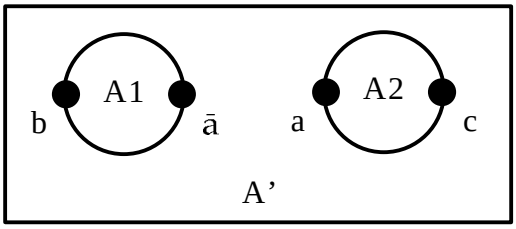
Un certain nombre d'opérateurs ont été définis pour construire des agents plus complexes à partir d'agents de base. L'opérateur '+' spécifie un choix entre deux agents. Dans l'exemple suivant, l'agent A adopte soit le comportement de A1, soit celui de A2.

A = A1 + A2

L'opérateur '|' spécifie la composition d'agents. Les deux agents ainsi liés évoluent en parallèle. Ils n'en forment plus qu'un au vu de l'extérieur. Toutes les actions de chacun des sous-agents peuvent être atteintes depuis l'extérieur. Dans l'exemple suivant A' est une composition de A1 et A2. Les actions ā de A1 et a de A2 peuvent donner lieu à une synchronisation, mais l'action ā de A1 peut également se synchroniser avec une action a extérieure.

A' = A1 | A2

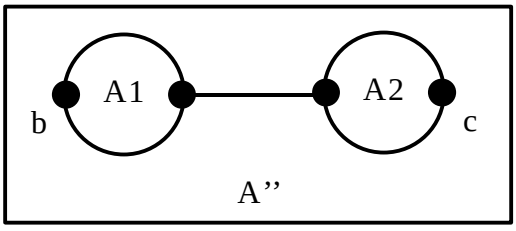
L'agent A' peut alors être représenté de la façon suivante :



L'ensemble des actions visibles depuis l'extérieur peut être réduit. Un opérateur de réduction est spécifié : '\'. Si on reprend l'exemple précédent, on peut restreindre l'agent A' aux actions b et c. Les actions a et ā deviennent ainsi invisibles depuis l'extérieur, ce qui assure qu'aucun autre agent ne pourra se connecter sur les ports a et ā, qui deviennent ainsi liés. Ces deux actions deviennent alors silencieuses et leur occurrence est indiquée par le symbole τ. Cette restriction se fait de la façon suivante :

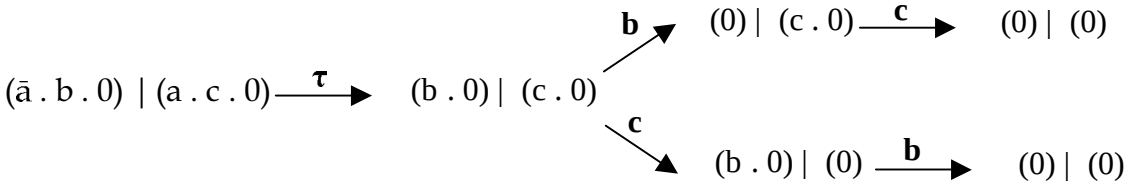
A'' = A1 | A2 \ {a}

L'agent A'' peut alors se représenter de la façon suivante :



Enfin, un dernier mécanisme est le renommage, qui consiste à affecter des nouveaux noms aux différentes actions d'un agent afin de pouvoir les réutiliser dans différents contextes. C'est d'autant plus important que le couplage d'une action émettrice avec une action réceptrice impose une unicité de nom.

Un agent peut en fait être défini comme un ensemble d'états. Le passage d'un état à l'autre se fait par l'intermédiaire d'une transition, qui correspond à une action. A partir de la spécification d'un agent on peut construire (en partie ou en totalité) son arbre de dérivation, qui décrit l'ensemble des exécutions possibles. Voici l'arbre de dérivation de l'agent A''.



A partir des concepts introduits précédemment, on peut définir le méta-modèle suivant (voir Figure 69). Un agent peut être un agent complexe, c'est-à-dire une composition ou une alternative entre 2 à n sous-agents, ou bien un agent simple, une séquence d'actions. Une action peut avoir un certain nombre d'actions complémentaires. Une action particulière est l'action de fin. Un agent définit un ensemble d'action dont certaines peuvent être cachées.

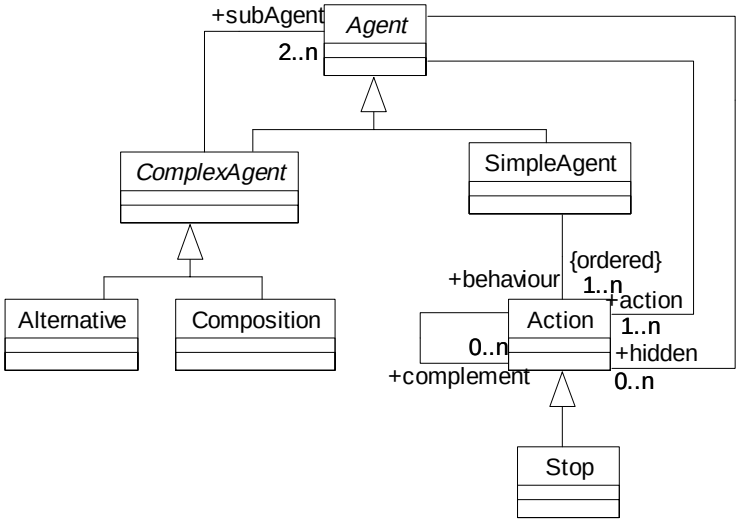


Figure 69. Méta-modèle (partiel) du modèle CCS

5.4. La sémantique d'actions

Il existe plusieurs approches permettant d'exprimer la sémantique d'un langage. Dans [59], Bertrand Meyer identifie les quatre possibilités suivantes :

- La sémantique traductionnelle, qui consiste à traduire les programmes écrits dans un langage vers un langage de base plus générique.
- La sémantique opérationnelle, qui consiste à spécifier les primitives d'un langage afin de les rendre exécutables par une machine virtuelle. Cette

spécification peut se faire en utilisant un langage de programmation ou une notation mathématique.

- La sémantique dénotationnelle, qui consiste à traduire les programmes écrits dans un langage vers une notation mathématique
- La sémantique axiomatique, qui s'attache à spécifier les effets visibles de chacune des primitives du langage (à l'aide d'axiomes et de règles d'inférence) plutôt qu'à définir la manière dont ces primitives sont effectivement exécutées.

Dans les travaux que nous présentons plus loin sur l'exécutabilité des modèles de processus, nous nous sommes plus particulièrement intéressés à la sémantique d'actions, qui est une branche de la sémantique opérationnelle. Les raisons qui ont orienté notre choix sont d'une part l'existence de travaux sur une sémantique d'actions pour les modèles UML et d'autre part l'existence des concepts d'opération dans le MOF et d'actions sémantiques dans les sNets, dont nous comptions tirer parti. Nous présentons dans cette section deux propositions autour de la sémantique d'actions, une pour les langages de programmation, et l'autre pour les modèles UML.

5.4.1. Action Semantics for Programming Language

Action Semantics for Programming Languages [62] (AS_PL) est une proposition de Peter Mosses visant à spécifier de façon claire et non ambiguë la manière dont s'exécute un programme écrit dans un langage de haut niveau. Les bénéfices attendus se situent à plusieurs moments du cycle de vie d'un langage de programmation. Au moment de la conception du langage, AS_PL peut servir de base de décision et de communication. Lors de l'implémentation d'un interpréteur ou d'un compilateur, AS_PL fournit des spécifications précises. Pour le programmeur, une parfaite connaissance des lois sous-jacentes au langage utilisé permet de valider et d'optimiser le code produit. Enfin des langages de programmation explicites peuvent être réutilisés pour en produire de nouveau, plus efficaces ou élégants.

AS_PL spécifie le comportement d'un langage de programmation en introduisant des actions sémantiques. Ces actions sémantiques vont agir sur les entités grammaticales définies dans la grammaire. Ainsi, si on a défini l'entité *Statement*, on peut définir la façon dont cette entité est exécutée grâce à la fonction *execute* qui prend une entité de type *Statement* en entrée. Ces actions peuvent avoir entre elles des liens de composition et d'ordonnancement, qui sont souvent en rapport avec les liens de composition et d'ordonnancement des entités grammaticales. Une même action sémantique peut se décliner différemment pour chacune des phrases correspondant l'entité grammaticale passée en paramètre. Ainsi une entité de type *Statement* s'exécute de façon différente selon qu'il s'agisse d'une conditionnelle, d'une boucle ou d'une affectation.

L'exemple suivant (Figure 70) est une grammaire proposée par Peter Mosses en exemple. Elle définit un *Statement* comme pouvant être l'affectation d'une expression à une variable (*Identifier*), une conditionnelle, une boucle ou une suite d'instructions.

```
grammar:
(1) Statement    = [[ Identifier ":=" Expression ]] |
                   [[ "if" Expression "then" Statement ( "else" Statement)? ]] |
                   [[ "while" Expression "do" Statement ]] |
                   [[ "begin" Statements "end"]].
(2) Statements   = [[ Statement ( ";" Statement )*]].
(3) Expression   = Numeral | Identifier | [[ "(" Expression ")"]] |
                   [[ Expression Operator Expression ]].
(4) Operator     = "+" | "-" | "*" | "<>" | "and" | "or".
(5) Numeral      = [[ digit+ ]].
(6) Identifier   = [[ letter (letter | digit)* ]].
closed.
```

Figure 70. Exemple de grammaire donnée par Mosses

Voici un exemple de programme (Figure 71) pouvant être conçu à partir du langage défini précédemment.

```
begin
        toto := 3 ;
        if ( toto <> 2 and toto <> 4 ) then
                toto := titi + 2
        else
                toto := 2 * 3 - 1 ;
        while toto <> 2 do
                toto := toto - 1
end
```

Figure 71. Exemple de programme

Peter Mosses a défini le comportement suivant à son langage (Figure 72). Un instruction (*Statement*) peut être exécutée (action *execute*). Si c'est une affectation, on commence par rechercher la zone mémoire allouée à l'identifiant (*Identier*) et évaluer (*evaluate*) l'expression (*Expression*) à affecter, puis on range le résultat du calcul dans la cellule. La démarche est bien sûr totalement différente si l'instruction est une conditionnelle, une boucle ou une suite d'instructions. De même l'évaluation d'une expression est différente selon qu'il s'agisse d'un numérique (*Numeral*), d'un identifiant, ou d'une opération sur deux expressions.

```
introduces: execute _ , evaluate _, the operation-result of _, the value of _ .
• execute _ :: Statements -> action [completing | diverging | storing] .
(1)   execute [[ I:Identifier ":=" E:Expression ]] =
            ( give the cell bound to I and evaluate E )
            then store the given number#2 in the given cell#1 .
(2)   execute [[ "if" E:Expression "then" S1:Statement ]] =
            evaluate E then
                  check the given truth-value and then execute S1
               or
                  check not the given truth-value and then complete .
(3)   execute [[ "if" E:Expression "then" S1:Statement  "else" S2:Statement ]] =
            evaluate E then
                  check the given truth-value and then execute S1
               or
                  check not the given truth-value and then execute S2 .
(4)   execute [[ "while" E:Expression "do" S:Statement ]] =
            unfolding
               evaluate E then
                  check the given truth-value and then execute S
                  and then unfold
                or
                  check not the given truth-value and then complete .
(5)   execute [[ "begin" S:Statements "end"]] = execute S.
(6)   execute [[ S1:Statement ";" S2:Statement]] = execute S1 and then execute S2.
• evaluate _ :: Expression -> action [giving a value].
(7)   evaluate N:Numeral = give the value of N.
(8)   evaluate I:Identifier =
            give the value bound to I or
            give the number stored in the cell bound to I .
(9)   evaluate [[ "(" E:Expression ")" ]] = evaluate E .
(10)  evaluate [[ E1:Expression O:Operator E2:Expression ]] =
            ( evaluate E1 and evaluate E2 ) then give the operation-result of O .
• the operation-result of _ :: Operator -> yielder [of a value] [using the given value²]
(11)  the operation_result of "+" = the number yielded by
            the sum of (the given number#1, the given number#2) .
(12)  the operation_result of "-" = the number yielded by
            the difference of (the given number#1, the given number#2) .
(13)  the operation_result of "*" = the number yielded by
            the product of (the given number#1, the given number#2) .
(14)  the operation_result of "<>" =
            not (the given value#1 is the given value#2) .
(15)  the operation_result of "and" =
            both of (the given truth-value#1, the given truth-value#2) .
(16)  the operation_result of "or" =
            either of (the given truth-value#1, the given truth-valuer#2) .
• the value of _ :: Numeral -> number .
(17)  the value of N:Numeral = number & decimal N .
```

Figure 72. Exemple d'actions sémantiques donné par Mosses

La définition des actions sémantiques se fait grâce à un langage mis au point par Mosses. Ce langage minimal introduit un certain nombre d'actions atomiques. Elles sont sensées pouvoir représenter tout ce qu'un langage de plus haut niveau peut vouloir exprimer. Elles permettent d'effectuer des calculs sur des valeurs, de gérer des

ordonnancements complexes entre actions, de rechercher et de mettre à jour des cellules mémoire, etc.

Le langage précédent peut aisément être réécrit sous la forme d'un méta-modèle sous le formalisme MOF (voir Figure 73).

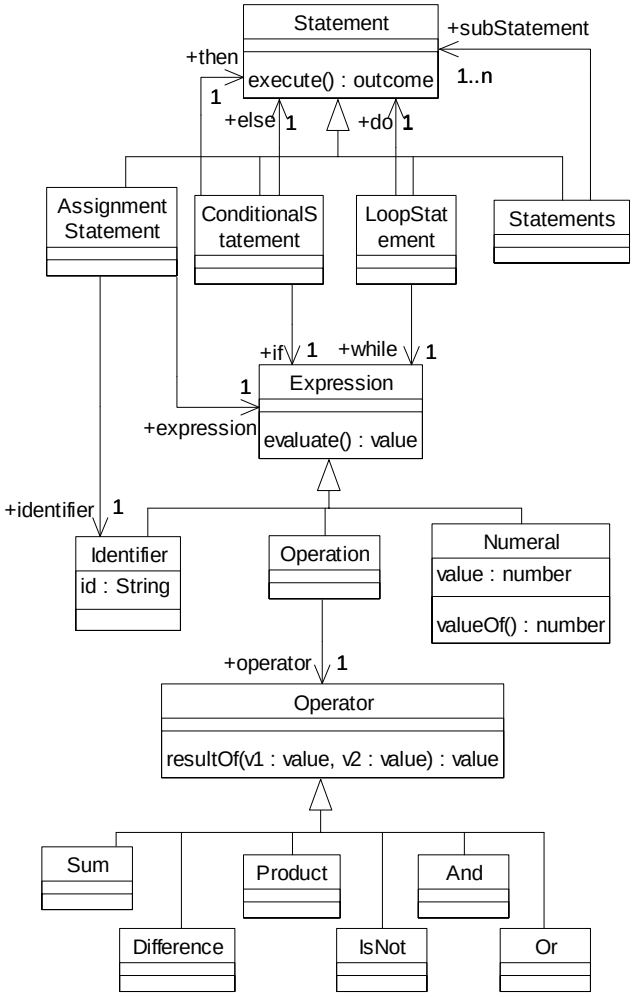


Figure 73. Définition de la syntaxe abstraite du langage comme un méta-modèle

Les différentes parties droites alternatives pour un même symbole dans la grammaire ont été définies à l'aide de relation d'héritage (*Statement* et *AssignmentStatement*). Les actions sémantiques sont devenues des opérations attachées à une entité. L'héritage et le polymorphisme vont permettre de redéfinir de façon élégante le contenu de ces méthodes pour chaque sous-classe d'une entité grammaticale. Ainsi, la méthode *execute()* définie au niveau de *Statement* sera redéfinie au niveau de chacune des sous-classes. La plupart des contraintes qui étaient uniquement exprimées de façon syntaxique (par exemple la multiplicité avec les symboles '+', '*' et '?') sont reprises au niveau du méta-modèle. Cela peut être fait en utilisant les contraintes de multiplicité sur les fins d'associations ou par l'utilisation de contraintes pré-définies (telles *ordered*) ou OCL. Un des seuls inconvénients du méta-modèle est qu'il est plus verbeux et qu'il peut nécessiter l'introduction d'entités

supplémentaires par rapport à la grammaire (par exemple l'entité *Operation* du méta-modèle). Cela est dû au fait que la définition de la grammaire peut être plus concise, une même entité grammaticale pouvant se traduire par plusieurs phrases différentes.

Si la traduction de la grammaire EBNF en méta-modèle MOF se fait relativement aisément, il n'en va pas de même pour les actions sémantiques. En effet, on peut spécifier les signatures des méthodes, mais il manque un langage permettant de spécifier leur contenu.

5.4.2. Action Semantics for UML

Action Semantics for UML (AS_UML) [68] est un travail en cours de finalisation à l'OMG. Son objectif est la définition d'un langage complémentaire d'UML pour la spécification précise de la sémantique d'exécution des modèles, mais il peut également être utilisé pour définir des transformations telles que l'application de patterns sur un modèle UML [94]. Un tel langage doit apporter un certain nombre de facilités pour :

- La simulation et la vérification des modèles,
- La génération de code, de jeux de tests, etc. [57]

AS_UML trouve donc naturellement sa place au sein du MDA. En effet, comme Sridhar Iyengar le signale, plus les modèles UML seront précis, plus le pourcentage de code généré sera important. Une des ambitions de AS_UML est d'être indépendant de toute plate-forme, cela afin de garantir la portabilité du modèle.

Dans les propositions initiales, AS_UML était divisée en deux parties :

- La définition de la sémantique d'exécution d'un modèle UML,
- L'extension d'UML par l'ajout d'un certain nombre d'actions.

La première partie a disparu du dernier document en date. Elle définissait le modèle d'exécution de différentes entités d'UML (voir Figure 74). L'exécution d'un modèle UML est vue comme un ensemble de clichés (*Snapshots*). Un cliché peut être rattaché à toute entité définie comme mutable, c'est-à-dire qui est susceptible de subir des évolutions au cours de son cycle de vie. Cette association est faite par le biais de l'identité (*Identity*) de l'entité mutable. Les entités mutables sont les classes, les objets, les associations et les liens. Le cliché d'une classe est un ensemble de valeurs de propriétés statiques (fins de liens et attributs) et d'instances. Le cliché d'une association est un ensemble de liens instance de cette association. Le cliché d'un objet est un ensemble de valeurs d'attributs et de liens. Il comprend également le lien vers le type dont il est l'instance. Le cliché d'un lien est vide (la vie d'un lien se résume à sa création puis à sa suppression). L'histoire (*History*) d'une entité mutable est donc une suite de changements menant d'un cliché à l'autre (*Change*). En plus des clichés reliés aux éléments mutables, AS_UML définit la notion de clichés d'exécution. Ceux-ci sont des états intermédiaires au cours d'un traitement. Un cliché d'exécution peut dépendre de plusieurs clichés. Les clichés d'exécution sont séparés par des étapes qui vont éventuellement provoquer des changements.

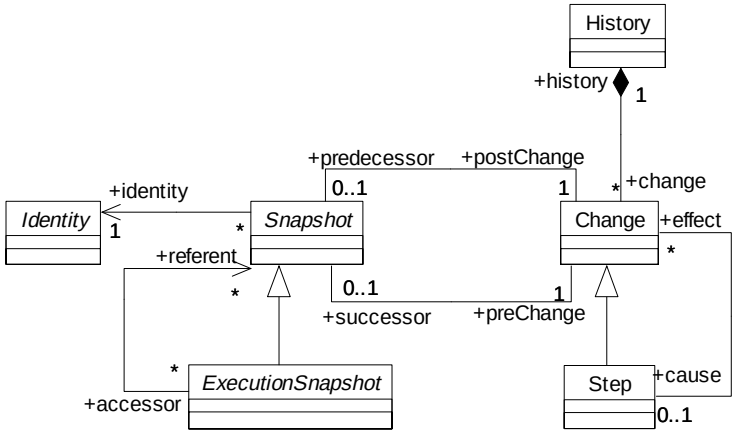


Figure 74. Le modèle d'exécution de toute entité mutable d'UML

La seconde partie (seule partie du dernier document) définit les actions. La spécification du comportement se fait grâce aux concepts de procédure et d'action. Une procédure encapsule une action et peut être utilisée pour définir le corps d'une méthode ou les traitements à effectuer à l'entrée ou à la sortie d'un état (voir Figure 75). Une action peut aussi bien définir une opération de mise à jour sur l'environnement que des opérations plus complexes.

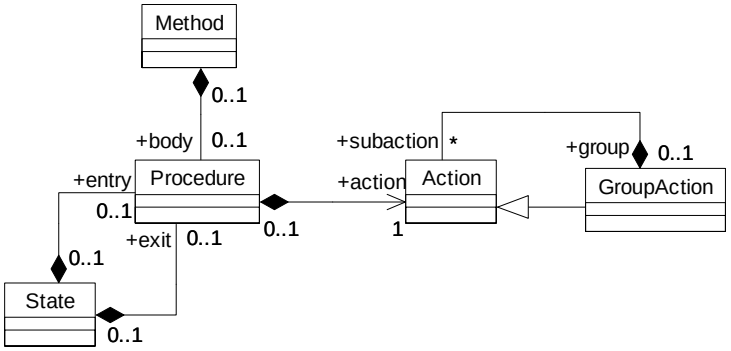


Figure 75. Intégration des actions au sein d'UML

Les actions, tout comme les procédures, vont prendre des paramètres en entrée (*InputPin*) et fournir des résultats en sortie (*OutputPin*) (voir Figure 76). L'ordonnancement entre actions peut se faire soit explicitement via le flux de contrôle, soit implicitement, via le flux de données. Dans ce dernier cas, une action ne peut démarrer que lorsque tous les paramètres attendus en entrée sont disponibles. Ceux-ci pouvant dépendre des résultats d'une autre action, on a ainsi un lien d'ordonnancement implicite entre les deux actions.

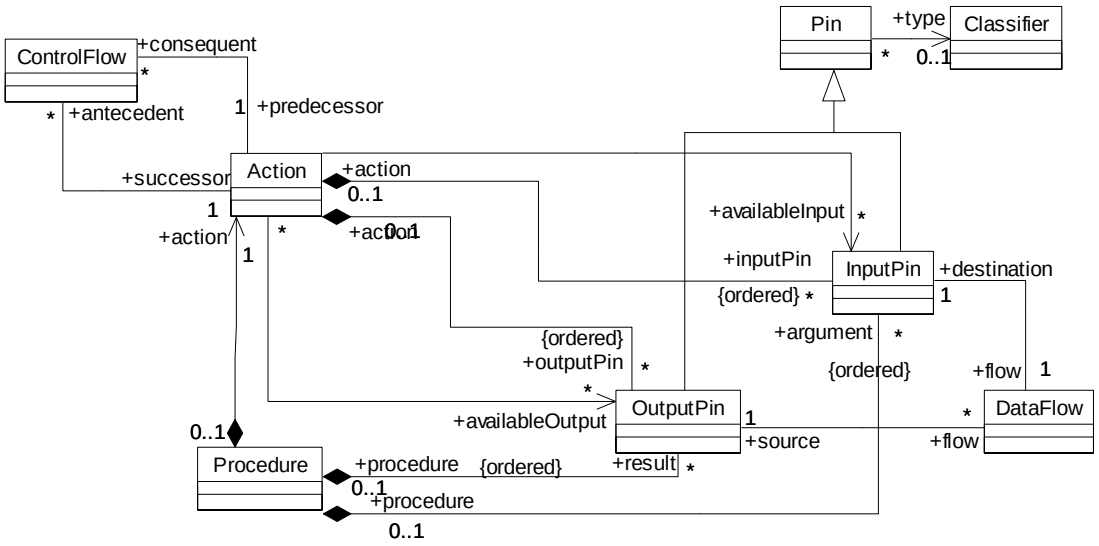


Figure 76. Définition des actions

Une typologie des actions a également été proposée. Les différents types d'actions identifiés sont :

- Les actions composées qui vont du simple regroupement d'actions aux structures de contrôle plus complexes (boucles, conditionnelles).
- Les actions de lecture écriture. On y trouve entre autres les actions gérant les objets (création, destruction, reclassification), les attributs (lecture, écriture) et les liens (création, destruction, accès au lien, accès à l'objet relié).
- Les actions de calcul permettant d'appliquer des fonctions primitives sur des éléments ainsi que des actions pouvant provoquer des effets de bord.
- Les actions sur collections définissant des mécanismes (filtre, itération, réduction) pour l'application d'actions sur des ensembles d'éléments.
- Les actions d'envoi et de réception de messages qui recouvrent à la fois les appels synchrones à des sous-procédures et les envois de signaux.
- Les actions d'exception permettant de lever des exceptions et des interruptions.

5.5. Conclusion

Dans bien des cas, on souhaite définir des modèles de processus exécutables. C'est particulièrement vrai lorsqu'on est dans les domaines du workflow ou de l'EAI. Un méta-modèle ne consistant qu'en un ensemble de concepts n'est alors pas forcément suffisant. Il reste encore à définir la manière dont un modèle sera interprété dans un contexte particulier. Pour répondre à cela, des algèbres de processus comme les modèles CSP et CCS ont été développées. Dans le domaine des langages de programmation, il existe également de nombreuses propositions de mécanismes de formalisation de la sémantique d'exécution. Parmi l'ensemble des travaux existants, nous avons sélectionné la sémantique d'actions

définie par Mosses, qui présente un pendant dans le monde UML avec Action Semantics for UML. Il nous reste donc à étudier la manière dont ces travaux peuvent être repris au sein de méta-modèles de processus. C'est ce que nous faisons dans la troisième partie de ce document.

Conclusion

Cette première partie introductive nous a permis de présenter les bases sur lesquelles s'appuient nos travaux. Ce premier travail de mise en correspondance du concept de processus utilisé dans différents domaines et différents formalismes nous amène déjà à établir un certain nombre de conclusions. Tout d'abord la méta-modélisation nous a permis de mettre en perspective un certain nombre de formalismes de description de processus de façon extrêmement simple et intelligible. Un des bénéfices majeurs de l'utilisation systématique de méta-modèles a été de faciliter la comparaison et l'alignement de formalismes aussi différents que TOVE, EDOC ou BPML. Cela nous a également permis de communiquer avec des personnes issues d'autres corporations. Ainsi, nous avons réalisé un méta-modèle dédié à l'ABC [11], méta-modèle qui a pu servir de support de discussion avec des gestionnaires. Cela n'aurait pas été forcément le cas si ce formalisme avait été exprimé en KIF ou en XML. Une autre conclusion importante est l'importance du concept de processus dans des domaines très variés. Sa définition varie selon les utilisations et les formalismes, mais on retrouve de nombreux points communs. Enfin, un méta-modèle est le plus souvent défini comme un ensemble de concepts, de relations, et d'assertions. On a vu que ce n'était pas forcément suffisant dans le domaine des processus qui sont des systèmes dynamiques destinés à représenter ou spécifier un ensemble d'exécutions. Si une proposition comme le MOF propose des mécanismes pertinents pour la représentation de systèmes, elle ne permet pas aujourd'hui de spécifier les règles régissant leur évolution.

Partie II
REALISATIONS INDUSTRIELLES

Introduction

Dans ce chapitre nous présentons un certain nombre de réalisations faites au sein de la société Sodifrance. Ces travaux concernent le développement ou l'adaptation d'outils dédiés à la description et à l'exploitation de modèles de processus et leur mise en œuvre, que ce soit sur un projet réel ou au travers d'un certain nombre de prototypes.

Nous commençons donc par détailler ce que nous avons appelé la chaîne d'ingénierie du processus. Il s'agit:

- D'un méta-modèle de processus correspondant aux besoins d'expression que nous avons pu identifier dans les documents méthode rédigés par les experts métier de Sodifrance
- D'un modeleur graphique permettant la définition de processus basés sur ce méta-modèle
- De mécanismes basés sur Scriptor-G et Scriptor-T afin de pouvoir réutiliser des référentiels de processus existants et alimenter des outils du marché, notamment des moteurs de workflow.

Cette chaîne d'ingénierie de processus a été mise en œuvre dans le cadre de nombreux projets et a fait l'objet de publications ([10], [12]). Nous étudions plus particulièrement son application au processus de tierce maintenance applicative déployé pour un groupe d'assurances. Nous présentons la démarche utilisée pour définir le processus en relation avec les utilisateurs et les experts métier, ainsi que le résultat obtenu et déployé sur l'environnement d'exécution de processus FlowMind de la société Akazi. Enfin, de nombreux prototypes ont été réalisés pour éprouver et améliorer ces différents outils. Nous terminons ce chapitre en en présentant un particulièrement significatif : l'intégration d'un outil de planification (MS-Project) pour gérer les délais et les affectations de ressources.

6. La chaîne d'ingénierie de processus

6.1. Introduction

Les attentes de la société Sodifrance concernaient initialement en priorité ses propres processus métier: migration de données, de plate-forme et tierce maintenance applicative. Sodifrance a développé une forte compétence dans ces domaines. De nombreux outils ont été réalisés pour répondre à des besoins particuliers à ces processus, particulièrement pour ce qui concerne l'analyse (de code, de système), et la transformation. On peut citer par exemple :

- Essor, un outil de rétro-ingénierie, qui permet de cartographier les différents composants déployés sur un mainframe (programmes Cobol, JCL, etc.) et d'établir les relations d'interdépendance
- Semantor, qui permet d'analyser du code Cobol et ainsi d'effectuer des études d'impact ou des audits de qualité.
- Data-Migrator, un outil dédié à la migration de données, qui permet de mettre en relation des structures de données d'un système source avec leurs correspondants dans le système cible, de définir les règles de transformation, et de générer le programme de migration.

Au fil des chantiers, Sodifrance a également acquis de l'expérience dans la conduite de projet. En plus de ces outils qui instrumentent chacune des tâches du processus, de nombreux documents existaient sur la manière de mener ce type de projet. Un certain nombre d'informations avaient été identifiées :

- le découpage en activités et en tâches,
- la distribution du travail à réaliser entre les rôles,
- les règles conditionnant l'ordonnancement entre les différentes étapes,
- les documents à produire, à utiliser,
- les outils pouvant assister dans la réalisation d'une tâche,
- etc.

Ces documents décrivaient les différentes méthodes en langage naturel. Il n'y avait donc pas de processus formalisé à strictement parler, mais un ensemble épars d'informations textuelles relativement homogènes dans leur contenu identifiant un certain nombre de pratiques éprouvées. Notre première tâche a donc été d'étudier comment ces différentes informations pouvaient être organisées afin de pouvoir être exploitées. Nous avons

commencé par faire une étude de différents outils du marché qu'on peut classer en deux types :

- les outils de modélisation de processus génériques,
- les outils de conception de workflow.

Parmi les premiers on peut citer des produits comme MEGA-Process. Ce dernier permet de décrire très simplement des modèles de processus. De plus il possède un véritable référentiel, permettant de réutiliser des informations entre plusieurs processus, et de réaliser des références croisées. Toutefois, il est principalement dédié à la génération d'un référentiel documentaire. Il ne présente donc pas toujours la rigueur et la précision nécessaires à la modélisation de processus exécutables. C'est particulièrement vrai en ce qui concerne les règles contraignant les transitions entre activités. Celles-ci sont la plupart du temps saisies dans un format textuel, ambigu, et sont donc difficilement exploitables.

Beaucoup d'éditeurs de moteur de workflow proposent également leur propre modeleur de processus. Ceux-ci présentent de nombreux inconvénients. Tout d'abord, ils sont souvent difficiles à mettre en œuvre pour des non-initiés. Or, nous souhaitions qu'un expert métier puisse facilement adapter le processus standard pour répondre à des besoins particuliers à un site. Ensuite, la définition des processus métier dans le formalisme fourni par un moteur de workflow pose des problèmes de pérennité. Chaque moteur de workflow introduisant des concepts particuliers, le processus est adapté à ce formalisme. Un certain nombre de patterns de workflow (au sens des patterns de workflow définis par van der Aalst [99]) sont appliqués de façon implicite pour contourner les manques de l'outil. Si l'on souhaite réutiliser le processus ainsi défini avec un autre moteur de workflow, il faudra alors extraire du modèle ce qui est générique et interpréter les constructions spécifiques. Il peut donc s'avérer extrêmement compliqué de passer d'un moteur de workflow à l'autre. Or, la plupart des processus étant déployés chez les clients, il peut se présenter le cas où celui-ci dispose déjà d'un moteur de workflow. De plus, un workflow s'exécute dans un environnement d'exécution particulier qui peut influer sur le modèle. Pour deux sites différents, le processus métier peut être identique, mais le workflow qui l'exécutera sera différent. Pour assurer une réutilisabilité maximale des processus métier, il est donc nécessaire de les préserver des particularités dues à la plate-forme d'exécution. Enfin, les formalismes proposés par les modeleurs de workflow sont extrêmement fermés. L'objectif étant de fournir un workflow exécutable, il est rare que des mécanismes d'extensibilité soit fournis.

Ce constat sur la difficulté de trouver un outil du marché répondant à l'ensemble de nos attentes en terme de précision, d'ouverture et de pérennité est à l'origine de nos travaux sur la chaîne d'ingénierie de processus. Nous commençons par présenter l'articulation des différents composants de cette chaîne. Puis nous présentons plus en détail chacun d'entre eux. Tout d'abord nous décrivons le méta-modèle que nous avons mis au point à partir des documents méthodologiques. Il constitue la colonne vertébrale de notre solution. Puis, nous donnons un aperçu de l'outil de définition de processus. Enfin, nous terminons en présentant les mécanismes d'ouverture, permettant d'exploiter les modèles ainsi spécifiés. En conclusion, nous esquissons quelques directions pour des travaux futurs.

6.2. Les composants de la chaîne

La chaîne d'ingénierie recouvre un certain nombre de composants permettant de définir des processus, de les transformer pour alimenter des outils du marché (autres modeleurs, outils de planification, moteurs de workflow, etc.), ou au contraire de récupérer des modèles pré-existants. Toute la solution que nous avons développée s'organise autour d'un méta-modèle de processus. Ce méta-modèle, PMM (Process Meta-Model), a été développé à partir des concepts que nous avons pu extraire des différents documents méthodologiques disponibles chez Sodifrance. Nous nous sommes également inspirés de travaux portant sur la représentation des processus (PIF, ARIS, WPDL, etc.). Ce méta-modèle, associé au format XML-sNets, constitue donc le bus d'interopérabilité entre les différents composants de la chaîne (Figure 77). Le modeleur graphique et Scriptor-G sont directement basés sur ce méta-modèle. Scriptor-T intègre PMM parmi l'ensemble des méta-modèles disponibles dans son référentiel. Ce dernier peut ainsi être le formalisme source ou cible d'une transformation.

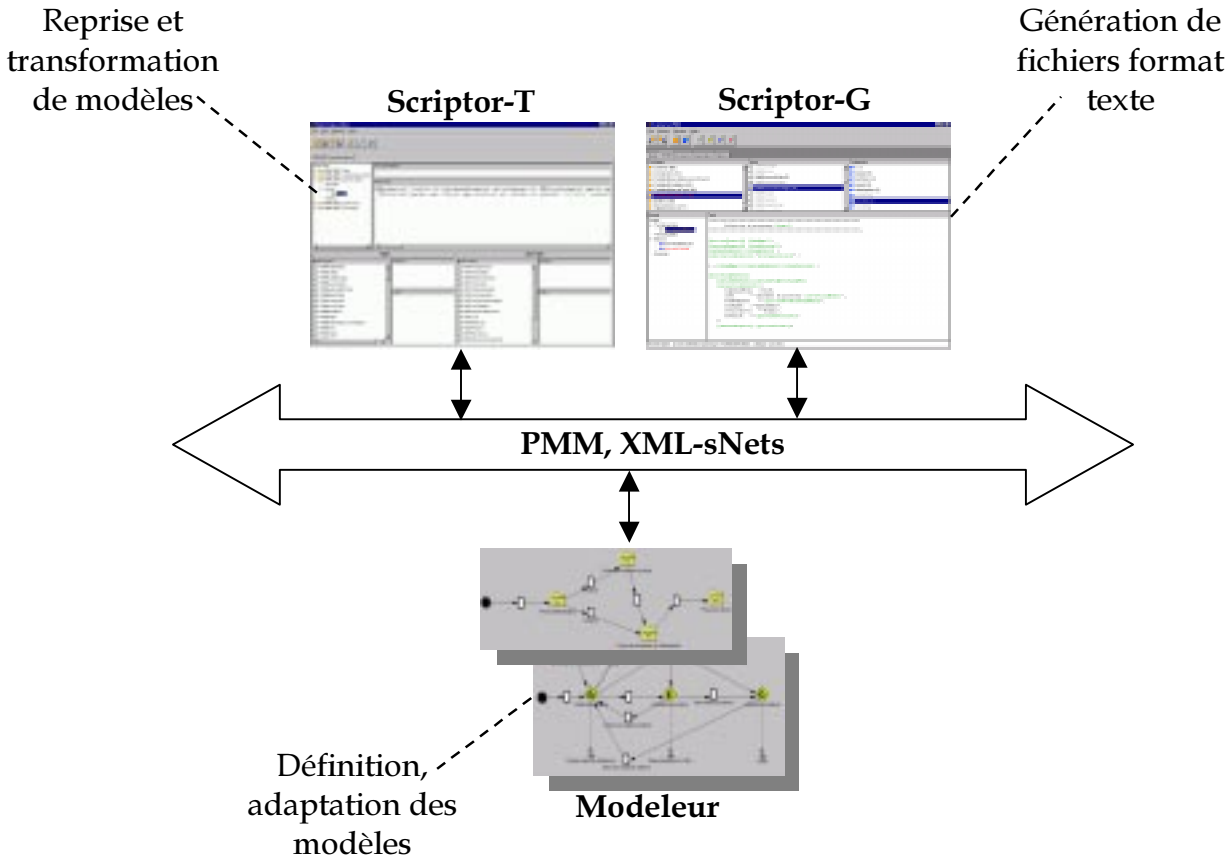


Figure 77. Le bus d'interopérabilité au coeur de la solution propriétaire de Sodifrance

La chaîne d'ingénierie de processus est donc constituée de trois composants, chacun ayant un rôle spécifique. Le modeleur permet de définir des modèles de processus. Ces modèles vont ensuite pouvoir être utilisés par Scriptor-G pour générer des fichiers textuels.

Au cours de nos différents travaux nous avons utilisé Scriptor-G pour produire des fichiers d'alimentation pour des moteurs de workflow, que ce soit les formulaires servant d'interface aux utilisateurs, le modèle du workflow ou encore de la documentation au format HTML. Scriptor-T pourra également les utiliser pour générer d'autres modèles. Nous nous sommes servis de ce dernier pour alimenter MS-Project à partir de modèles définis grâce au modeleur. Enfin, Scriptor-T permet également la reprise de modèle. Nous avons pu expérimenter cette fonctionnalité en récupérant des modèles de processus depuis MEGA-Process ou MS-Project. Ces modèles ainsi récupérés peuvent alors être retravaillés dans le modeleur (on va par exemple préciser les règles de transitions ou les rôles responsables de la réalisation de chacune des tâches), puis être acheminés vers Scriptor-G pour produire un workflow exécutable.

6.3. Le méta-modèle

6.3.1. L'organisation du méta-modèle

Ce méta-modèle a été initialement développé pour les besoins d'expression spécifiques aux métiers de Sodifrance. Il a principalement été construit à partir du vocabulaire utilisé par les experts métier, tout en s'inspirant de nombreux travaux menés tant dans le milieu universitaire (PIF) que dans l'industrie (ARIS). Ce méta-modèle a été défini en utilisant le formalisme des réseaux sémantiques (sNets). Toutefois il n'y a aucun obstacle à l'exprimer en utilisant le formalisme MOF. C'est d'ailleurs cette représentation qui est utilisée. Nous présentons ici une version simplifiée de ce méta-modèle, certains aspects ayant été occultés afin d'alléger le propos de considérations secondaires.

Au plus bas niveau un processus n'est qu'un ensemble de tâches ordonnées en fonction de règles indiquant des dépendances ou des synchronisations. Mais c'est aussi bien plus que cela, c'est un vecteur d'intégration entre les différentes dimensions d'une entreprise. En effet les tâches du processus vont être réalisées par des personnes physiques ou par des équipes, des outils vont être utilisés pour assister l'exécution du travail, et des produits, intermédiaires ou finaux (voir Figure 78), vont être consultés, modifiés ou créés.

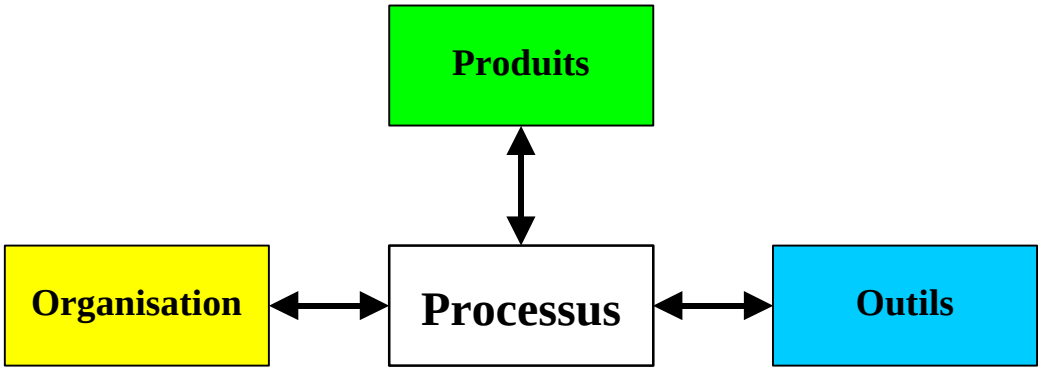


Figure 78. Intégration des composantes d'une entreprise par les processus

Pour respecter cette décomposition en quatre dimensions (organisation, outils, produits, processus), nous avons découpé notre méta-modèle en un certain nombre de paquetages (voir Figure 79). *PMMRootDefinition* est le paquetage de plus haut niveau, il contient les entités de bases du méta-modèle (telle *PMMEntity* dont vont hériter toutes les entités du méta-modèle et qui définit que toute entité a un nom et une documentation). Au niveau directement inférieur on trouve cinq paquetages. *PMMProductsDefinition*, *PMMToolsDefinition*, *PMMOrganisationDefinition* et *PMMProcessDefinition* qui définissent chacun une des composantes de l'entreprise (respectivement les produits, les outils, l'organisation et les processus). Le cinquième paquetage, *PMMExtensionsDefinition*, introduit les mécanismes d'extension permettant de créer virtuellement de nouvelles méta-entités. Enfin, au plus bas niveau, *PMMIntegratedProcessDefinition* met en relation tous ces paquetages.

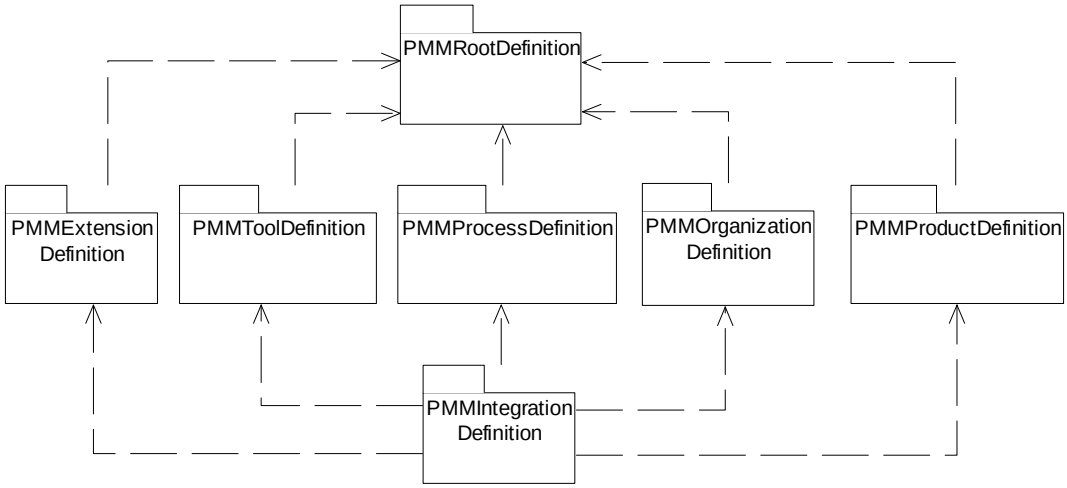


Figure 79. Organisation du méta-modèle PMM

6.3.2. Les mécanismes de modularité

Dans notre méta-modèle nous avons deux entités permettant d'organiser les modèles de façon modulaire : les paquetages (*PMMPackage*) et les processus (*PMMProcessPackage*), le processus étant un sous-type du paquetage (Figure 80).

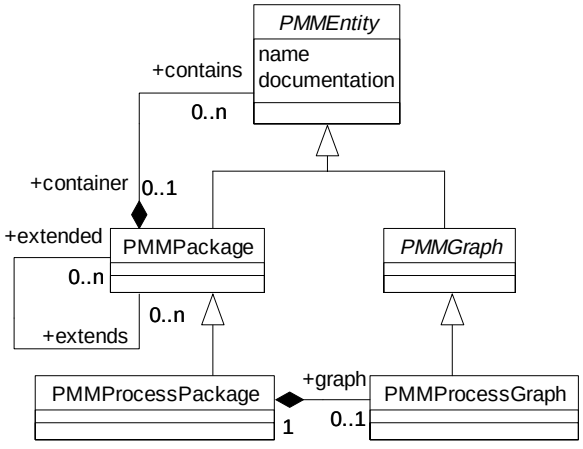


Figure 80. Paquetages et processus dans le méta-modèle PMM

Les paquetages définissent des conteneurs dans lesquels on va pouvoir stocker des entités statiques : des produits, des données, des rôles, des groupes, des outils, etc. Ces entités pourront alors être réutilisées dans les paquetages et les processus étendant le paquetage conteneur.

Un processus est un sous-type de paquetage, c'est donc également un conteneur. De plus, il définit un graphe d'activités (*PMMProcessGraph*, un graphe étant un ensemble d'étapes et de transitions, voir Figure 81). Tout comme un paquetage, un processus permet de stocker des entités statiques. Mais, en plus de ces entités, il contient un enchaînement d'étapes définissant les tâches à réaliser.

Une autre différence entre processus et paquetage tient au fait qu'un processus ne peut être étendu, ni par un paquetage, ni par un autre processus. Cette contrainte peut être exprimée en OCL de la façon suivante :

```
context PMMProcessPackage inv:
        self.extended->isEmpty();
```

6.3.3. Définition d'un processus

Le graphe (*PMMGraph*) est l'entité qui définit le découpage d'un processus (dans le cas d'un *PMMProcessGraph*) en étapes (*PMMStep*), ces étapes étant ordonnancées par des transitions (*PMMTransition*) (voir Figure 81). Ces étapes peuvent être des pseudo-étapes (*PMMPseudoStep*), correspondant à des étapes utilitaires telles que le point de départ (*PMMBeginStep*). Initialement, nous n'avons pas défini de point de fin dans un graphe car nous considérons que l'exécution d'un graphe est terminée quand toutes les étapes et transitions formant le graphe sont inactives. Les étapes peuvent être également des étapes d'action (*PMMActionStep*), qui sont soit des activités (*PMMActivity*), soit des tâches (*PMMTask*).

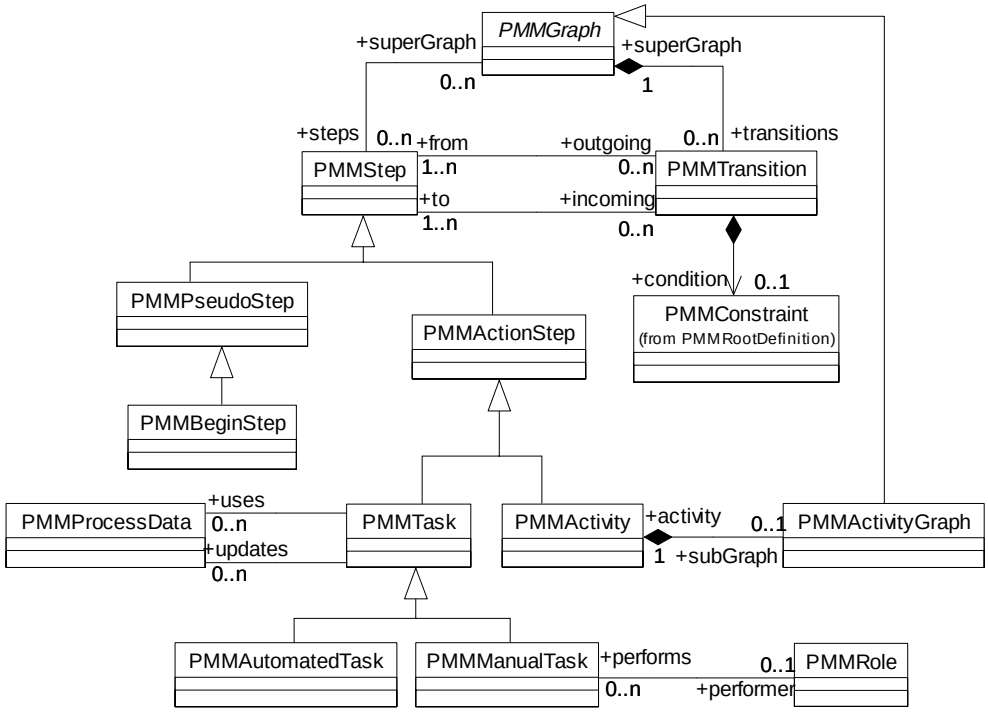


Figure 81. Définition d'un processus dans le méta-modèle Sodifrance

Une activité définit en fait un sous-graphe (*PMMActivityGraph*). C'est donc une partie complexe du processus. La tâche est l'élément atomique du processus. Elle met à jour et utilise des données (*PMMProcessData*). Elle peut être manuelle (*PMMManualTask*) si elle demande l'intervention d'un agent humain (quelle que soit l'ampleur de cette intervention) ou automatique (*PMMAutomatedTask*) si elle est complètement automatisée. Le lancement d'une transaction batch par un agent humain est une tâche manuelle. Elle ne sera automatisée qu'à partir du moment ou ce lancement pourra être réalisé sans aucune intervention humaine (soit par un moteur de workflow, soit par le système d'exploitation, etc). La tâche manuelle n'est pas directement reliée avec la personne ou le groupe de personnes chargées de sa réalisation. On passe par l'intermédiaire d'un rôle (*PMMRole*). La notion de rôle est contextuelle au processus. Ainsi dans un processus de validation, on aura souvent une négociation entre deux rôles, celui qui soumet une requête et celui qui valide. Ce processus peut devenir un pattern qu'on pourra implémenter dans différents contextes. Selon le contexte les deux rôles peuvent être affectés à des structures organisationnelles différentes. Enfin la transition (*PMMTransition*) définit la règle de passage d'une étape à l'autre (ou d'un groupe d'étapes vers un autre). Quand toutes les étapes antérieures sont terminées, la contrainte de passage (*PMMConstraint*) est examinée. Si elle est avérée alors les étapes ultérieures peuvent être exécutées. Sinon cette branche du processus est coupée. Les contraintes de passage portent sur les données du processus.

6.3.4. Intégration d'un processus

Une fois qu'on a défini le processus et un environnement d'exécution (organisation, produits, outils), il reste à les mettre en relation. C'est ce qui est fait dans le paquetage

PMMIntegratedProcessDefinition qui établit les liens existant entre un processus et les composantes statiques constituant l'environnement (voir Figure 82).

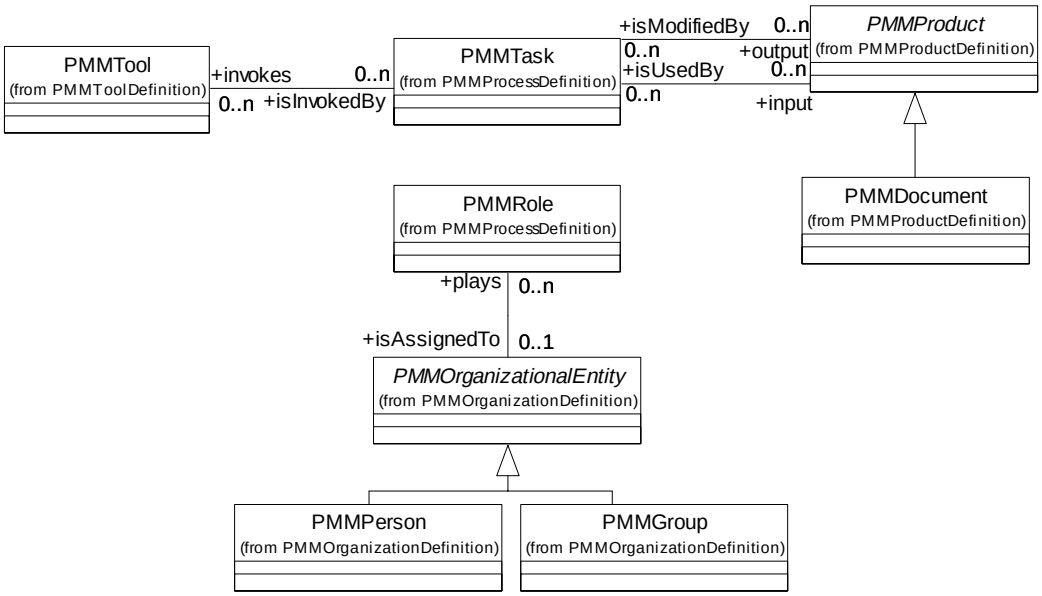


Figure 82. Intégration d'un processus dans le méta-modèle Sodifrance

Ainsi la réalisation d'une tâche peut nécessiter d'avoir accès des produits (*PMMProduct*, défini dans le paquetage *PMMProductsDefinition*). Elle peut également en créer ou en modifier. Pour mener à bien une tâche, il est possible d'utiliser des outils (*PMMTool* défini dans *PMMToolsDefinition*). Enfin, un rôle est lié à une entité organisationnelle (*PMMOrganisationalEntity* défini dans *PMMOrganisationDefinition*). Cette entité organisationnelle peut aussi bien être une personne qu'un groupe de personnes. Un groupe peut être une équipe (un groupe de personnes travaillant sur un même projet), un groupe de compétences (les personnes sachant programmer en Java), un groupe hiérarchique (les chefs de projet), etc.

6.3.5. Mécanismes d'extension

En plus des entités dédiées à la modélisation de processus nous avons défini des mécanismes d'extension. Ces mécanismes sont au nombre de trois:

- Les tagged values (*PMMTaggedValue*)
- Les entités indéfinies (*PMMUndefinedEntity*)
- Les associations indéfinies (*PMMUndefinedAssociation*)

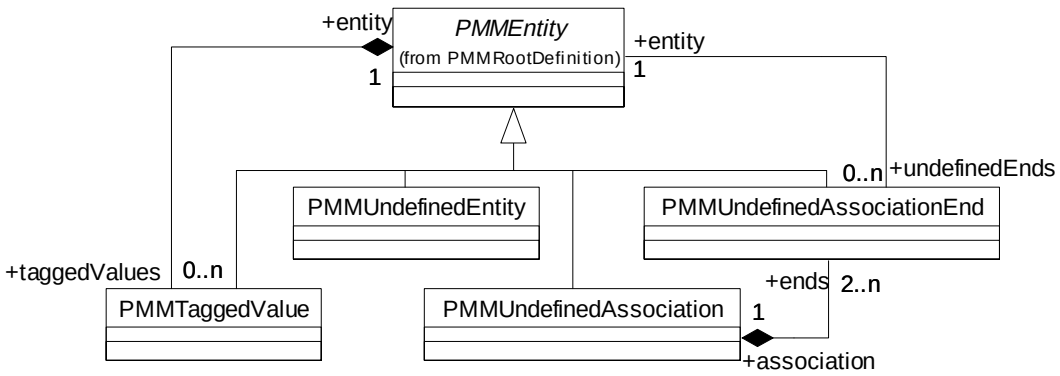


Figure 83. Mécanismes d'extension dans le méta-modèle Sodifrance

Le mécanisme des tagged values est similaire à celui qui est défini dans UML. Les tagged values permettent simplement d'étendre une méta-entité au niveau du modèle en lui rajoutant un nouvel attribut. Ainsi, l'ajout d'une tagged value « averageTime » sur une activité reviendrait à enrichir le méta-modèle avec un nouveau concept héritant de l'activité et définissant un temps moyen de réalisation.

Les entités indéfinies et les relations indéfinies permettent d'ajouter virtuellement une nouvelle entité au méta-modèle. Elles ont deux attributs : *metaType*, le type virtuel de l'entité, et *metaPackage*, le paquetage contenant ce type. Ainsi déclarer une entité indéfinie qui a pour *metaType* « MyType » et pour *metaPackage* « MyPackage » revient à modifier le méta-modèle en rajoutant un nouveau paquetage appelé « MyPackage » qui étend *PMMRootDefinition* et qui contient l'entité « MyType », qui spécialise *PMMEntity*. Ce nouveau paquetage « MyPackage » est ajouté dans la liste des paquetages étendus par *PMMIntegratedProcessDefinition*. Le mécanisme est pratiquement similaire pour la création d'une relation indéfinie, mis à part le fait que cette nouvelle association spécialise *PMMAssociation* et que le paquetage dans lequel elle est sensée être définie doit étendre les paquetages contenant les entités liées par cette nouvelle association.

Les nouvelles méta-entités ainsi définies n'ont pas d'existence réelle au regard du méta-modèle et ne peuvent donc être sujettes à des contraintes de cohérence. Elles n'ont de signification que pour un utilisateur externe et averti (cet utilisateur pouvant par exemple être un outil de génération qui incorpore dans ses règles la gestion des entités indéfinies de *metaType* « MyType »). Elles sont particulièrement utilisées pour enrichir le modèle selon la cible de génération sélectionnée. En effet, ce méta-modèle n'intégrant pas les spécificités pouvant exister dans tel ou tel moteur de workflow du marché, il est souvent nécessaire d'utiliser ces mécanismes afin d'enrichir un modèle pour qu'il puisse prendre en compte des informations manquantes.

6.4. Le modeleur graphique

Notre objectif en définissant un méta-modèle était de fournir un formalisme permettant aux experts métier de Sodifrance d'expliciter des processus. Pour faciliter leur description, ainsi que leur communication au sein de l'entreprise, nous avons réalisé un outil de modélisation graphique [91]. Il a été complètement développé en Java. Au cœur de cette application on trouve la sMachine, c'est-à-dire une implémentation du moteur sNets. Ainsi, le modeleur est basé sur un méta-méta-modèle, ce qui lui assure une grande flexibilité. L'introduction de nouveaux concepts ne nécessite pas un effort de codage trop important. De plus, on profite ainsi des mécanismes standards des sNets, tels que l'import/export au format XML-sNets. Pour faciliter la manipulation des modèles, cette couche sNets a été masquée par une couche définissant les concepts définis dans le méta-modèle. De cette façon on ne manipule pas des *Nodes* et des *Links*, mais des *Tasks*, des *Transitions* et des *Roles*. Cette couche ne restreint pas la flexibilité du formalisme car elle est entièrement générée à partir du méta-modèle. Au-dessus de cette couche intermédiaire se place la couche graphique. Une entité graphique encapsule une entité du méta-modèle. Cette couche graphique a été développée à partir de la bibliothèque graphique jKit/GO, distribuée par la société Instantiation (http://www.instantiations.com/go/download.asp).

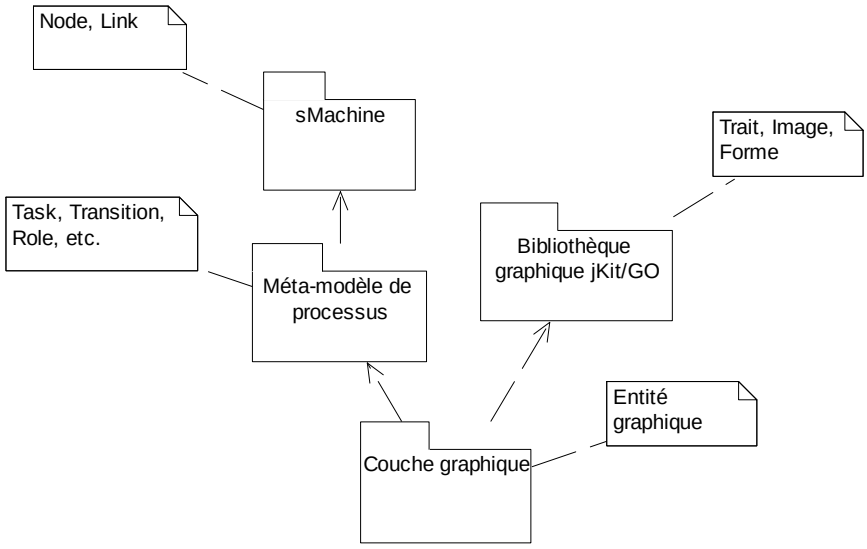


Figure 84. Architecture du modeleur

Le modeleur reprend l'ensemble des concepts définis dans le méta-modèle. La première vue permet de spécifier l'organisation des paquetages et des processus (Figure 85).

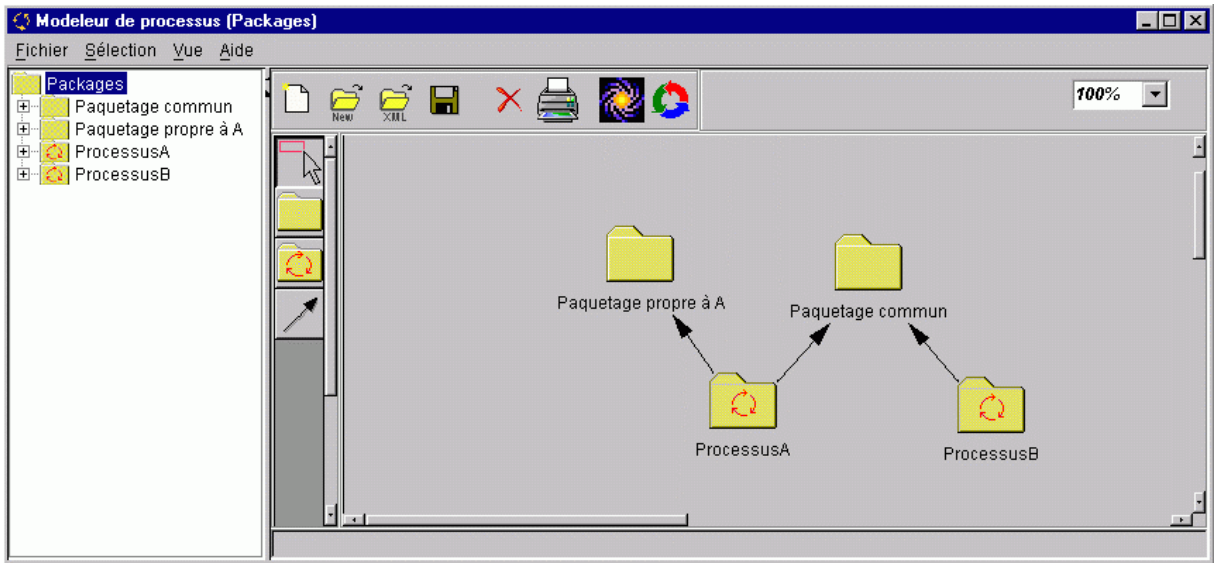


Figure 85. Définition de paquetages et de processus

Un processus contient des tâches et des activités, activités qui peuvent également être décomposées en tâches et activités. La navigation dans un processus se fait donc par un système de vues, une vue décrivant le contenu d'un paquetage, d'un processus ou d'une activité (Figure 86).

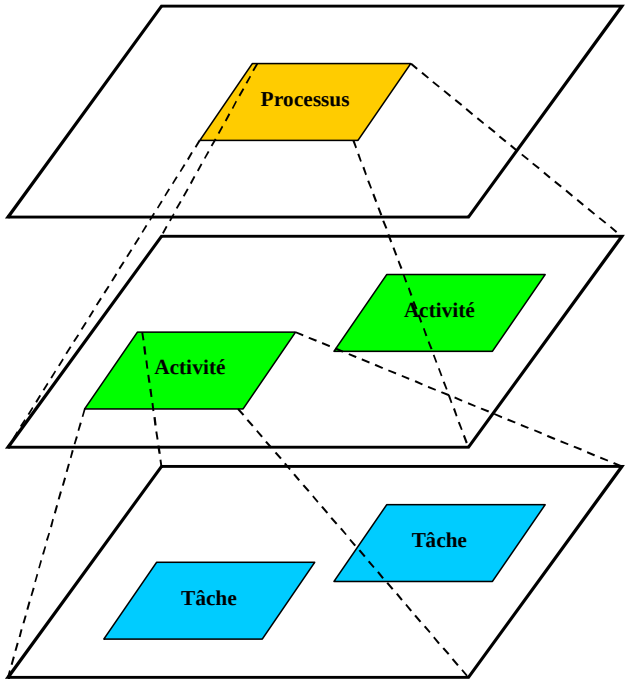


Figure 86. Organisation arborescente des vues

On peut voir sur la Figure 87 la façon dont sont représentés quelques-uns des concepts du méta-modèle parmi les plus significatifs.

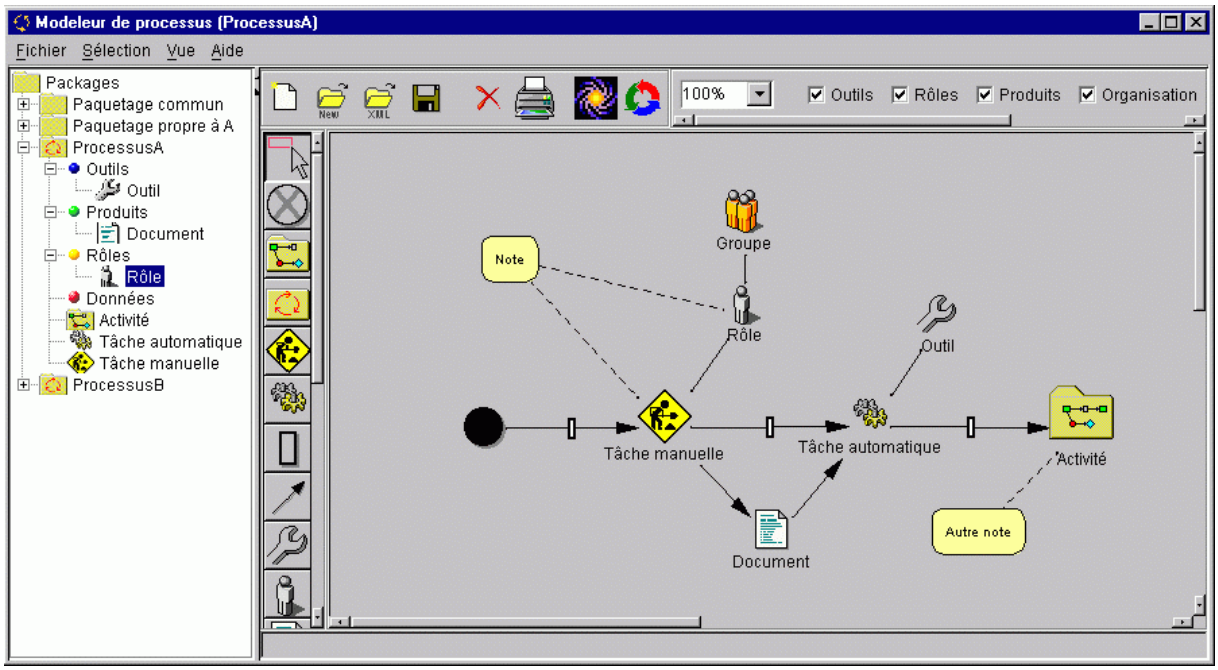


Figure 87. Représentation graphique de quelques concepts importants du méta-modèle

Un processus est principalement composé d'un ensemble d'activités et de tâches. Autour de ce squelette viennent se greffer tous les éléments annexes nécessaires à la prise en compte de l'environnement (outils, documents, rôles, etc). Un processus assez complexe peut donc vite devenir difficilement lisible. C'est pour répondre à ce problème qu'a été implémenté le mécanisme de filtrage. Les filtres permettent de masquer certaines catégories d'entités afin d'alléger la représentation graphique. La pose d'un filtre se fait en cochant une des cases à cocher situées en haut à droite de l'espace de travail. Toutes les entités appartenant à la catégorie, ainsi que les liens reliant ses entités, deviennent alors invisibles. Dans l'exemple suivant (Figure 88) nous pouvons voir tout d'abord l'espace de travail où seuls les rôles ont été conservés visibles. Les groupes, outils et documents disparaissent alors de l'espace de travail. La définition des catégories et la répartition des entités du méta-modèle entre elles sont paramétrables.

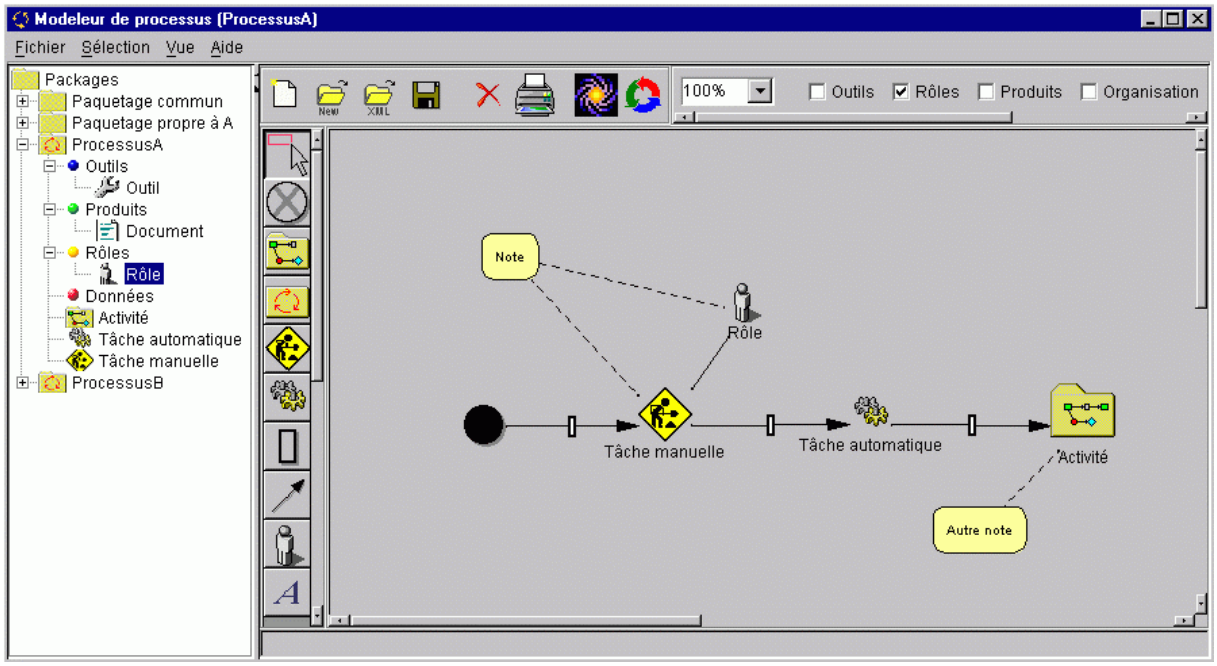


Figure 88. Utilisation des fonctionnalités de filtrage

L'ordonnancement entre étapes est spécifié à l'aide des transitions. Les deux exemples suivants (Figure 89 et Figure 90) montrent comment sont représentés:

- Les chemins conditionnels,
- Les boucles,
- Les points de parallélisme (ou fourches),
- Les points de synchronisation (ou jointures).

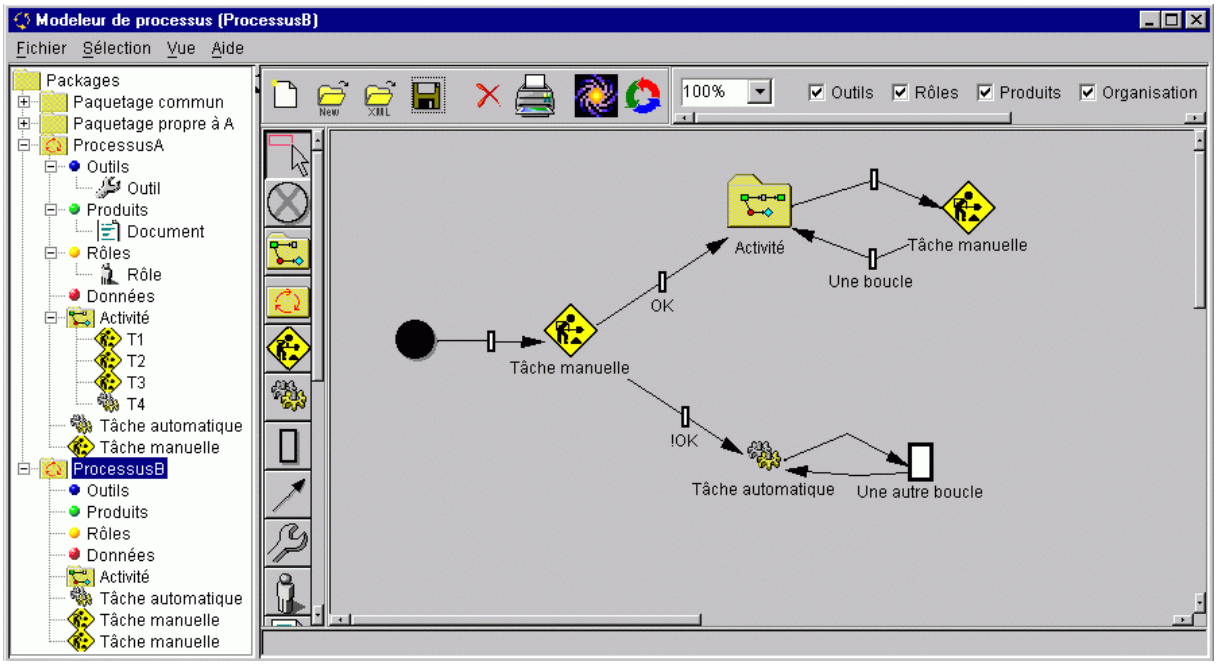


Figure 89. Chemin conditionnel et boucles

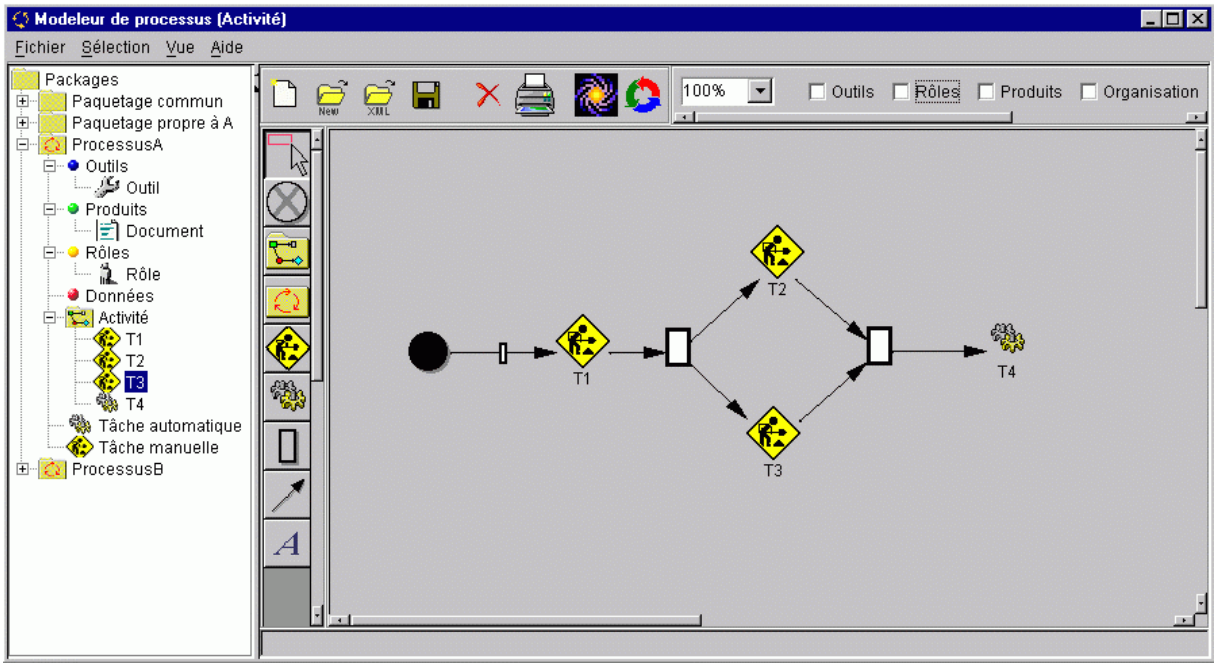


Figure 90. Fourche et jointure

Un des principaux inconvénients des modeleurs de processus génériques est qu'ils ne fournissent pas de syntaxe formelle pour la saisie des contraintes. Pour éviter ce travers, nous avons défini un langage extrêmement simple, permettant de définir des contraintes sur les données du processus. L'éditeur de contraintes est un assistant graphique dont le but est de faciliter la saisie des contraintes. Le principe de fonctionnement de cet assistant est de permettre à l'utilisateur de construire des contraintes sans avoir à écrire les opérateurs, mots clés ou variables de ce langage. Ces éléments peuvent être ajoutés par un simple « clic » de la souris sur les boutons disposés sous la zone de saisie. Lorsqu'on clique sur un des boutons situés sous la zone de saisie de la contrainte, le texte correspondant à ce bouton s'insère à partir de la position courante du curseur. Si du texte est déjà sélectionné, ce texte sera supprimé et remplacé par le texte inséré.

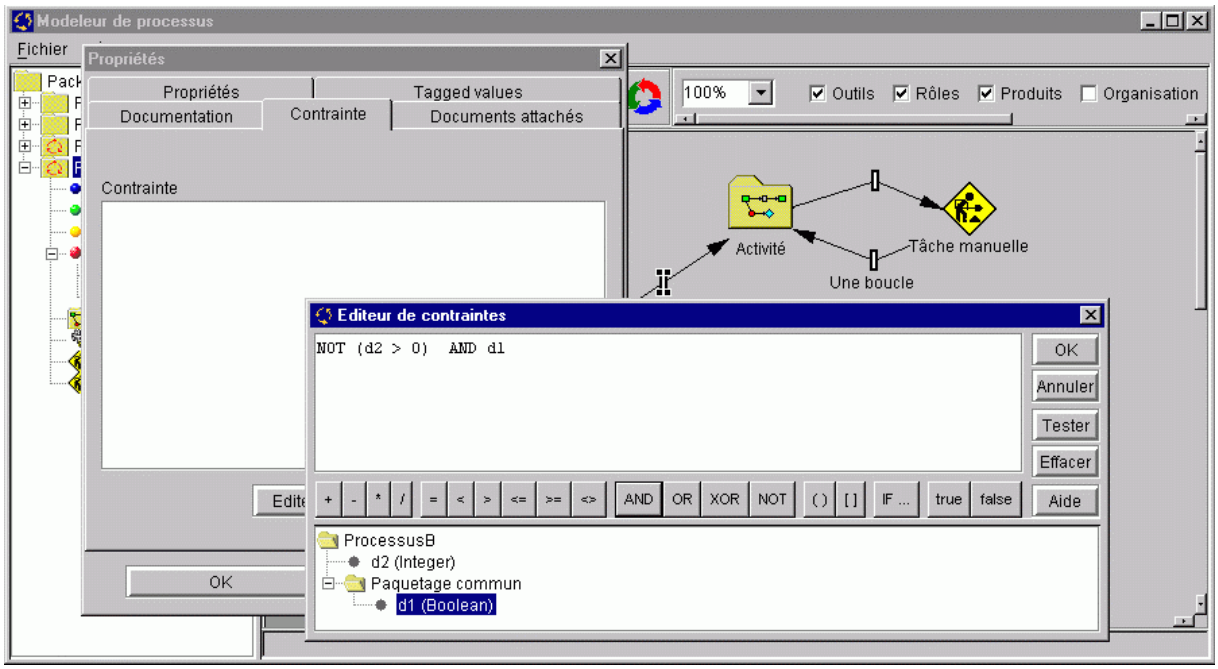


Figure 91. L'éditeur de contraintes

On a vu précédemment les différents mécanismes d'extensibilité définis dans le méta-modèle, tagged values, entités et relations indéfinies. Les tagged values sont accessibles depuis l'onglet « tagged value » des propriétés de chaque entité (Figure 92). Un mécanisme a été implémenté pour attacher des tagged values aux entités du méta-modèle. En effet, lorsqu'on démarre un nouveau projet il est courant de commencer par définir des tagged values sur les différentes entités. Elles sont souvent fonction de la cible de génération choisie. La prédéfinition de tagged values permet de ne pas avoir à les re-saisir systématiquement. Pour cela, il suffit de spécifier dans un fichier de configuration les noms des tagged values et d'indiquer l'entité du méta-modèle à laquelle elles sont rattachées. Il est également possible de définir des valeurs par défaut, ainsi que des ensembles de valeurs.

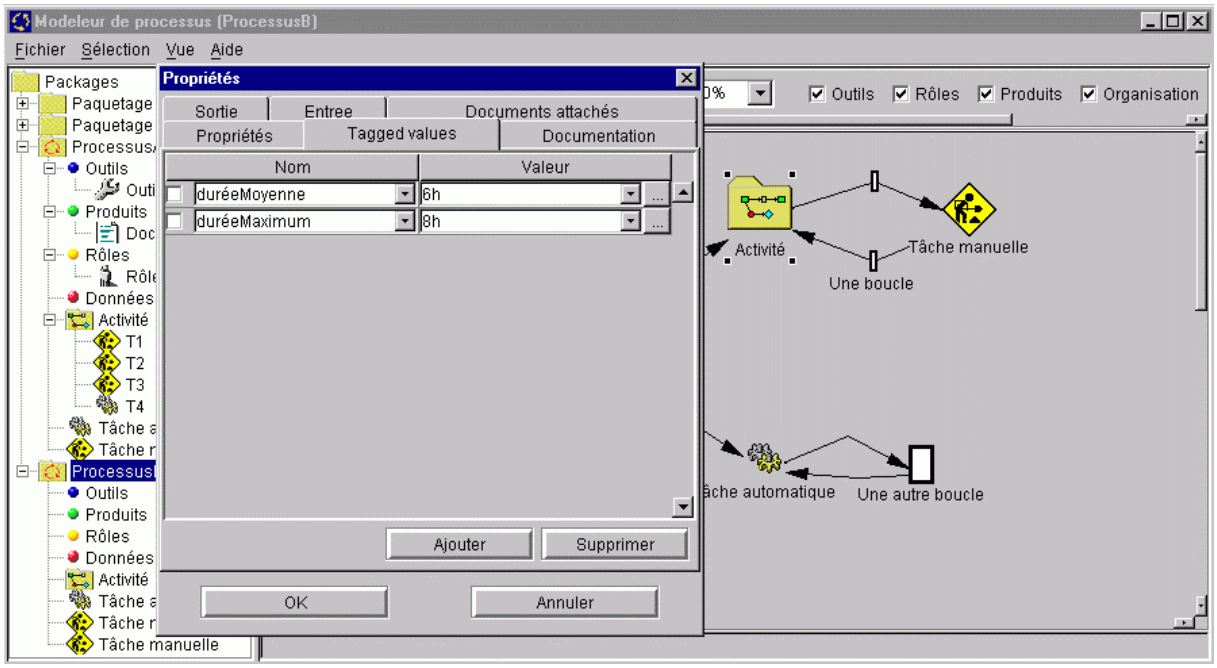


Figure 92. Définition de tagged values

Les entités et relations indéfinies sont définies de la façon suivante (Figure 93 et Figure 94).

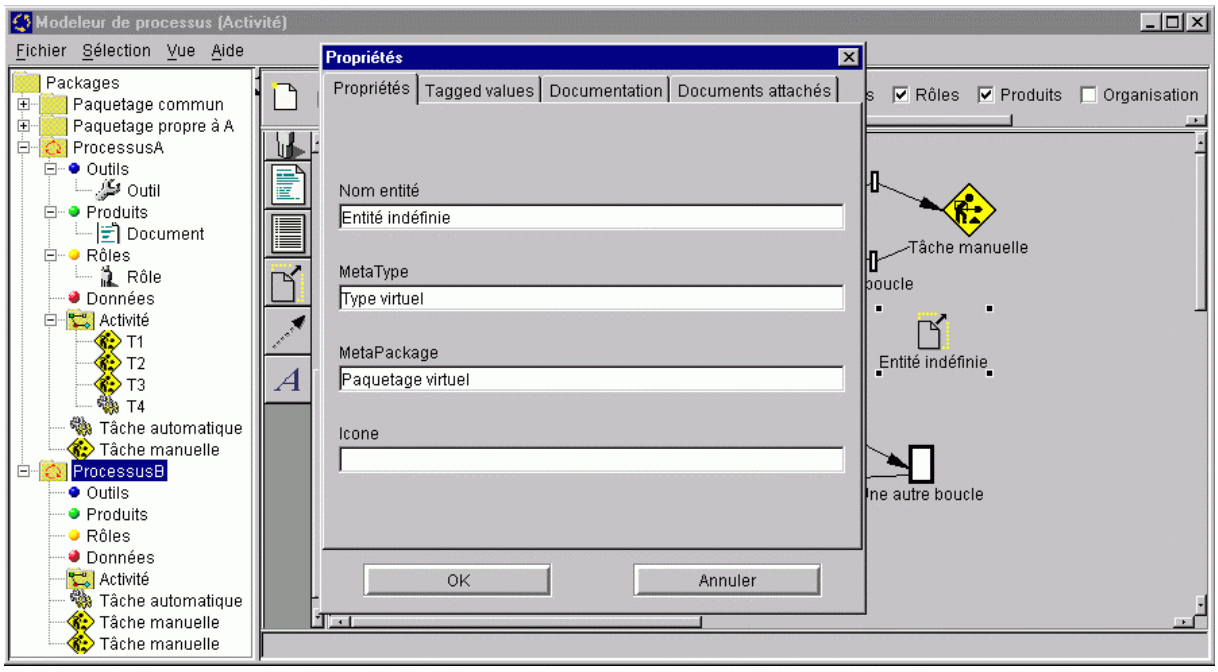


Figure 93. Définition d'une entité indéfinie

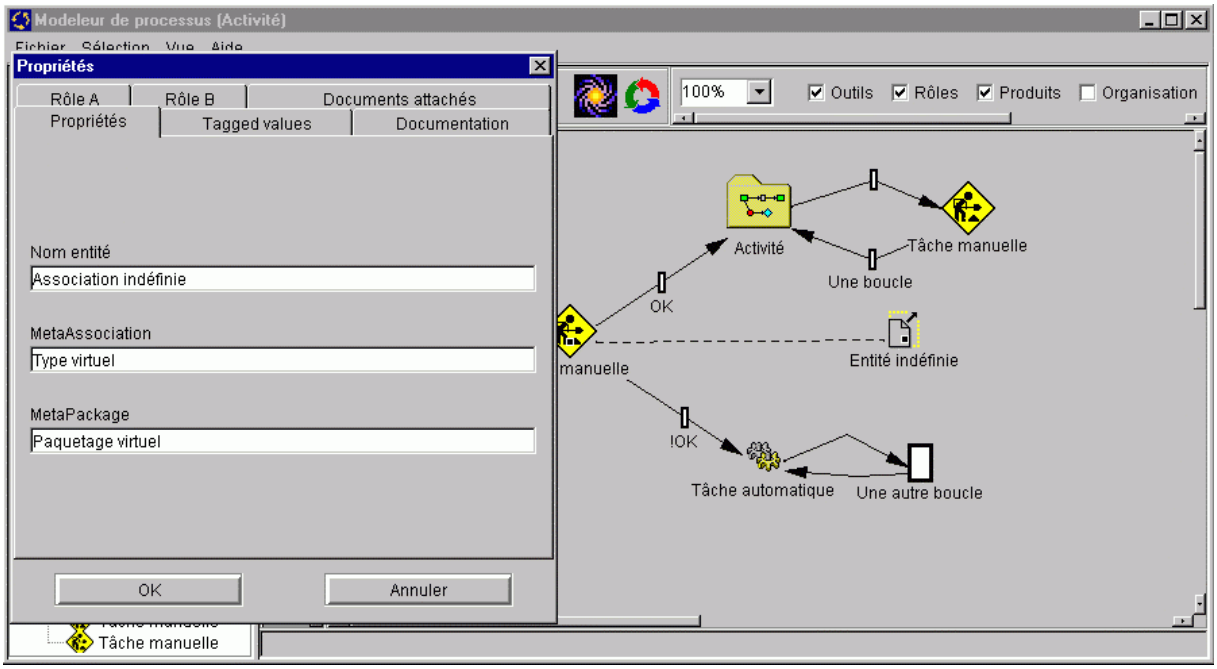


Figure 94. Définition d'une relation indéfinie

Enfin, deux autres fonctionnalités importantes ont été implémentées. Il y a tout d'abord la reprise de modèles non graphiques. En effet, les modèles provenant du transformateur de modèle ne contiennent pas d'informations graphiques. Toutefois, il est fréquent que l'on doive les retravailler dans le modeleur. Ces modèles peuvent être intégrés dans le modeleur, leur contenu étant placé arbitrairement sur le plan de travail. Enfin, Scriptor-T peut être invoqué directement depuis le modeleur. Ces deux outils partagent les mêmes classes de manipulation du méta-modèle. Le transfert d'information d'une application à l'autre se fait alors directement, et ne nécessite donc pas l'import/export d'un fichier XML-sNets.

6.5. Les mécanismes d'ouverture

Notre but en développant un méta-modèle de processus et le modeleur associé n'était pas seulement de fournir aux experts métier de Sodifrance un outil de représentation. Nous avions aussi la volonté d'exploiter ces modèles de façon automatisée pour alimenter différents outils du marché (moteurs de workflow, outils de planification, etc.). Bien sûr, de nombreux environnements de workflow proposent leur propre formalisme et un couplage fort entre le modèle et son implémentation [29]. L'implémentation est directement générée à partir du modèle ou ce dernier est interprété. Toutefois, ces modèles restent trop proches de l'implémentation. Ils y perdent ainsi en compréhensibilité, en maintenabilité, en réutilisabilité et en pérennité. De plus, ces systèmes de gestion de processus ne fournissent pas forcément les mécanismes d'ouverture permettant de réutiliser leurs modèles pour alimenter d'autres outils.

D'autre part, pour les cas où des processus ont déjà été formalisés, nous souhaitions être en capacité de les reprendre. Le méta-modèle de processus que nous avons défini devient ainsi un méta-modèle pivot, permettant le passage d'un outil à l'autre (Figure 95). Cette stratégie a ainsi été mise en œuvre pour alimenter un moteur de workflow (FlowMind de la société Akazi) à partir d'un outil de modélisation de processus du marché (MEGA-Process).

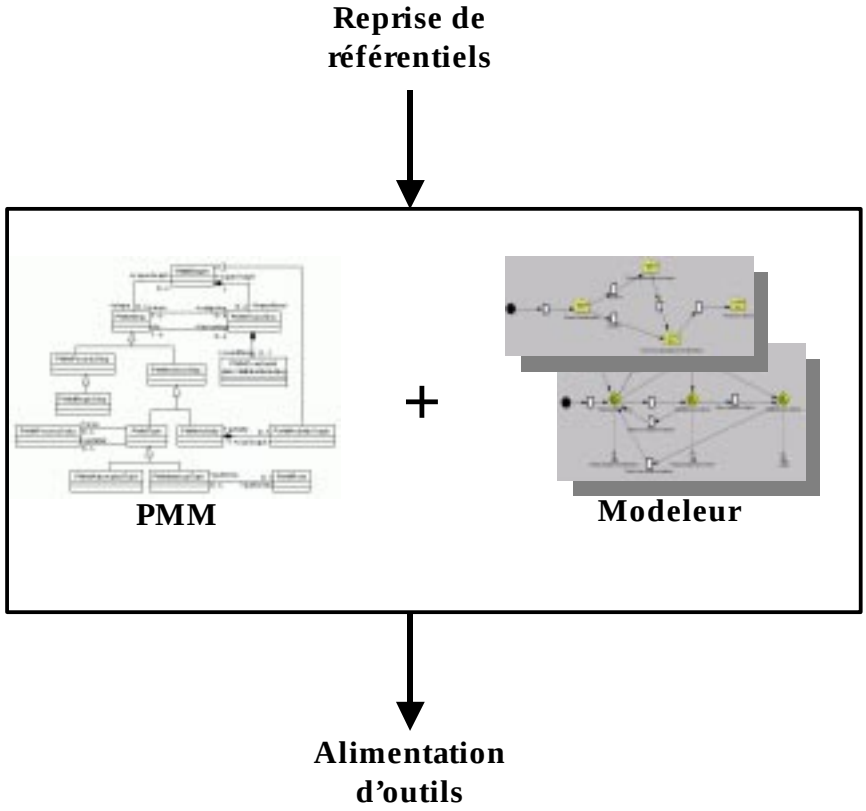


Figure 95. Le méta-modèle PMM comme un format pivot

Pour cela nous avons utilisé deux outils différents : ScriptorG et Scriptor-T. Scriptor-G est dédié à un méta-modèle. Scriptor-T est plus générique puisqu'il peut embarquer plusieurs méta-modèles différents sous la forme de composants Java. Dans les deux cas le travail de développement de l'outil a été minime puisqu'il est possible de générer la version de Scriptor-G dédiée à un méta-modèle particulier, tout comme le composant Java de gestion de ce méta-modèle destiné à Scriptor-T, à partir d'une description MOF de ce méta-modèle. Nous avons donc pu axer notre travail sur la partie concernant l'écriture des scripts de génération ou des règles de transformation permettant de passer d'un formalisme à l'autre.

Les avantages de cette approche par génération et/ou transformation sont nombreux. Tout d'abord on évite les phases de re-saisie manuelle des informations communes entre deux environnements de définition de processus. On peut donc ainsi obtenir un gain de temps non négligeable. De plus, le risque de voir apparaître des divergences entre modèles est plus limité. Comme le passage d'un formalisme à l'autre est automatisé, les informations ne sont pas altérées.

Un autre avantage apporté par ces techniques de transformation et de génération est qu'elles supportent une approche itérative par prototypage. Les mises à jour sur le modèle de plus haut niveau d'abstraction peuvent être rapidement et aisément intégrées dans le modèle exécutable. Cela facilite la mise au point d'un processus en collaboration avec des experts métier. En effet, la validation d'un processus sur une description statique n'est pas toujours chose aisée. Avoir la possibilité de générer rapidement un processus exécutable présente alors un avantage considérable. On peut ainsi non plus valider le modèle, mais son implémentation en train de s'exécuter. Et cela, sans prendre le risque d'un effet tunnel entre le moment de la livraison des spécifications et celui de la livraison du prototype.

Enfin, un avantage parmi les plus importants est la possibilité d'appliquer des patterns de transformation de façon systématique. Ainsi, on va pouvoir:

- Adapter le processus à son environnement d'exécution
- Définir des règles génériques pour contourner les limites d'un formalisme
- Prendre en compte des normes (pour la qualité, la charte graphique)

Dans le premier cas, on peut citer l'exemple d'un prototype de processus de développement logiciel que nous avons développé pour un client. Dans ce contexte, les affectations des rôles d'un processus lors de l'exécution étaient fournies par une application externe de planification. S'il arrivait qu'aucune affectation n'ait été définie pour un rôle, le responsable du projet devait en être averti. Enfin, l'application externe de planification devait être avertie lorsqu'une tâche était démarrée ou terminée. Ainsi, pour chaque tâche manuelle définie dans le modeleur, on avait dans le workflow la séquence d'activité suivante (Figure 96).

Dans le modeleur	**Dans le moteur de workflow**
	1. L'outil de planification est averti que la tâche est démarrée. 2. On recherche le réalisateur affecté au rôle responsable de la réalisation de la tâche. 3. Si aucun réalisateur n'est affecté, on demande au chef de projet d'en choisir un. 4. Lorsqu'un réalisateur est affecté, on lui signale qu'il a la tâche à réaliser 5. L'outil de planification est averti que la tâche est terminée.

Figure 96. Un exemple de pattern pour prendre en compte la spécificité de l'environnement d'exécution

De cette manière, le modèle de processus reste générique. Il n'est pas pollué par des préoccupations spécifiques à l'environnement dans lequel vont s'exécuter ses instances. Cette prise en compte de l'environnement est faite au moment de la génération. De plus elle

est systématique. Ce pattern est appliqué à l'ensemble des tâches manuelles, sans risque d'oubli.

Dans le deuxième cas on peut citer l'exemple du moteur de workflow FlowMind de la société Akazi (que nous présentons plus en détail dans la section suivante). Une des restrictions du formalisme développé par Akazi est que tout processus, ou étape composée, débute par une et une seule étape. Ainsi, il se pose un problème pour les cas où les premières étapes d'un graphe d'activités sont des étapes s'exécutant en parallèle. Pour régler ce problème nous avons décidé d'introduire automatiquement une étape vide en début de graphe, à chaque fois que le graphe du modèle source démarre par des tâches en parallèles.

Enfin, ces services de génération peuvent évoluer tout au long du cycle de vie du développement du processus. Selon l'avancement du modèle, on utilisera un ensemble de scripts spécifiques permettant de valider un aspect particulier du modèle. Ainsi, dans les phases amont de la mise au point d'un modèle de processus, le paramétrage des applications invoquées par les tâches automatiques ne constitue pas une priorité. L'accent est plutôt porté sur le découpage et l'ordonnancement des tâches, leur affectation ou la validité des règles de transition. Pour permettre au modélisateur de positionner les tâches automatiques, tout en en faisant une description incomplète, nous avons développé des services de génération permettant de remplacer ces tâches automatiques par des tâches manuelles. Celles-ci sont affectées à un rôle pré-défini (le système). On peut visualiser les données qui leur sont acheminées, et saisir les données qu'aurait du renvoyer l'application. Ainsi, le processus peut-il s'exécuter dans son ensemble, et être validé, et cela bien que le modèle d'origine soit incomplet. Un tel mécanisme avait déjà été développé dans l'environnement de représentation et d'exécution de processus défini par Walt Scacchi, l'Articulator [83]. Cet environnement est basé sur un moteur de règles. Une différence par rapport à notre approche est que le moteur a été conçu de façon à exécuter des processus même incomplets, c'est-à-dire dont la décomposition en tâches n'a pas été complètement spécifiée. Dans notre approche, les systèmes d'exécution utilisés n'acceptent que des processus complets. C'est la génération qui se charge de transformer le modèle incomplet en processus exécutable, en laissant de côté certains aspects ou en les précisant de façon arbitraire.

6.6. Conclusion

Dans cette section nous avons présenté la chaîne d'ingénierie de processus, une solution industrielle développée au sein de Sodifrance. Cette solution propose des mécanismes de représentation de processus et de transformation basés sur des techniques de méta-modélisation. A l'origine, ces travaux avaient pour but de fournir aux experts métier de Sodifrance un environnement leur permettant de décrire des processus. Des expérimentations ont en particulier était faites sur les processus de développement logiciel, de maintenance, et de tierce maintenance applicative (voir la section suivante).

Toutefois, il semble que cette solution soit assez générique pour être appliquée à d'autres domaines. Ainsi, une expérimentation a été réalisée en milieu bancaire sur des processus d'après-vente (déclaration de vol d'une carte, etc.). De plus, il a été également validé que la solution pouvait s'adapter dans de nombreux contextes. Comme nous le montrerons dans les sections suivantes, de nombreux types d'artefacts différents ont pu être produits à partir des modèles de processus (workflow, formulaires, documentation, etc.). Concernant les workflows, des essais ont été faits avec différents moteurs pour tester les possibilités d'interconnexion. Nous avons pu valider la démarche sur les trois moteurs de workflow que nous avons expérimenté : W4, Domino Workflow et FlowMind.

Un contrat de partenariat technique a d'ailleurs été conclu avec la société Akazi, éditrice de la solution FlowMind. Cette solution était dotée d'un outil de conception de workflow très technique. En effet, un workflow FlowMind est une succession d'invocations de composants Java. Malgré une interface conviviale, les concepts manipulés étaient difficiles d'accès pour nombre d'utilisateurs. Un des objectifs du partenariat était donc d'enrichir la solution FlowMind avec un outil de modélisation plus adapté à des corporations non informaticiennes. Dans le cadre de ce partenariat, nous avons organisé un transfert des connaissances et des technologies développées dans notre solution. Nous avons également participé à la spécification de ce nouvel outil, principalement en ce qui concerne la définition du méta-modèle.

Cette solution est donc aujourd'hui opérationnelle. Néanmoins, de nombreux points restent à approfondir. Les deux suivants nous semblent particulièrement importants :

- la réutilisabilité
- l'ouverture vers d'autres méta-modèles.

La solution telle qu'elle existe aujourd'hui limite la réutilisabilité aux entités statiques (documents, outils, groupes, etc.) contenus dans des paquetages. Par contre, nous n'avons pas défini de mécanismes permettant de réutiliser des entités dynamiques (activités et tâches). Ayant détecté ce problème, nous nous sommes heurtés à la contrainte qui a été évoquée lors de l'étude de SPEM, l'unification. Il y a nécessité d'établir des correspondances entre les entités statiques du processus et ceux de l'activité qu'on intègre. Dans notre méta-modèle cette notion d'entité statique recouvre les documents, rôles, outils, groupes, données, etc. On peut bien sûr imaginer de nombreux moyens pour résoudre ce problème. Par exemple un dialecte XML de type XPath ou la partie d'OCL dédiée à la navigation dans le modèle pourraient très bien être utilisé pour définir les correspondances entre les entités du processus et celles de la tâche. Toutefois, ces langages peuvent être difficiles d'accès pour des non informaticiens. Nous avons donc laissé cette piste en suspens pour des recherches ultérieures.

Un certain nombre de concepts définis dans notre méta-modèle le sont de façon très minimale. On peut citer entre autres les outils, les produits, les entités organisationnelles ou encore les données. Il nous apparaît donc très pertinent de s'intéresser au couplage du méta-modèle de processus avec des méta-modèles décrivant d'autres domaines. On peut bien sûr citer UML pour la définition des données du processus. On peut également citer les

travaux actuellement en cours à l'OMG sur la définition d'un modèle de structure organisationnelle [64]. Ce point a fait l'objet de quelques travaux que nous présentons par la suite.

7. Une première mise en oeuvre industrielle

7.1. Introduction

Le premier champ d'expérimentation de la chaîne d'ingénierie de processus a été une tierce maintenance applicative gérée par la société Sodifrance pour un groupe d'assurances. Le processus était défini dans un certain nombre de guides méthodologiques et dans le plan qualité. Toutefois, un certain nombre de problèmes avaient été identifiés. Tout d'abord, même si le processus était relativement bien décrit, sa mise en œuvre et son contrôle présentait de sérieuses lacunes. Il était très difficile de s'assurer que chaque intervention était bien réalisée conformément au processus établi. Cela dépendait uniquement de la bonne volonté des collaborateurs, et de la bonne compréhension qu'ils avaient du processus. Dans le meilleur des cas ils avaient eu accès aux documents méthodologiques et s'en servaient comme guide. Dans le pire des cas ils les ignoraient, et réinventaient leur processus au quotidien. Cette dernière situation n'est pas sans poser de sérieux problèmes quand la qualité et les délais du service rendu sont fixés contractuellement. De plus, l'accueil de nouveaux collaborateurs sur la plate-forme de maintenance avait un certain coût : le temps de familiarisation avec le processus.

Un autre exemple de problème typique à l'exécution est celui de la re-saisie. Les informations n'étaient pas acheminées automatiquement entre applications et documents. Il n'était donc pas rare qu'un collaborateur doive les re-saisir, soit pour renseigner un champ, soit pour paramétrer l'invocation d'une application. Cette re-saisie était faite au mieux par l'utilisation du « couper-coller », mais elle pouvait également être réalisée de manière totalement manuelle. Cette dernière méthode implique bien sûr de sérieux risques de déviations ainsi qu'une importante perte de traçabilité.

Enfin, le contrôle de l'avancement des processus était réalisé au travers de tableaux de bord réalisés à partir des informations collectées auprès des différents acteurs. Chacun d'entre eux avait à remplir un certain nombre de documents précisant l'état d'avancement des processus sous sa responsabilité. Cette saisie étant uniquement dépendante de leur bon vouloir, les oublis étaient fréquents. Disposer d'un état d'avancement fiable se révélait alors être extrêmement difficile pour chaque chef de projet. Cette dimension du suivi est d'autant plus importante que la relation avec le client est étroite. L'équipe chargée de la maintenance d'un site doit pouvoir être en mesure de répondre à tout moment à un client s'informant de l'état d'avancement d'un processus de correction.

Guider les collaborateurs, fluidifier les échanges d'information et fournir un suivi à jour sans imposer une charge supplémentaire de travail, sont autant de raisons qui ont motivé l'introduction d'un moteur de workflow au sein de la plate-forme. Une rapide étude des moteurs de workflow du marché nous a conduit à choisir le moteur de workflow FlowMind. En effet, celui-ci présentait de nombreux avantages en terme de déploiement (client léger), d'ouverture (facilité d'intégrer des composants Java), et d'alimentation (un format d'import textuel était disponible). Enfin le coût a également fait partie de nos critères de choix.

Dans cette partie nous allons relater la façon dont nous avons mis en œuvre la chaîne d'ingénierie de processus, afin de modéliser et de générer le processus exécutable. Nous commençons par présenter ce qu'est une tierce maintenance applicative. Puis, nous détaillons le processus tel qu'il a été défini pour ce site particulier. Nous étudions ensuite la démarche que nous avons adoptée pour la production du processus automatisé. Nous terminons par présenter FlowMind ainsi que le workflow déployé. La conclusion résume les différents constats qui ont pu être tirés de cette application industrielle.

7.2. La tierce maintenance applicative (TMA)

Une tierce maintenance applicative consiste à sous-traiter la maintenance de son parc logiciel à une société tierce. Dans notre cadre d'application, le travail est confié à Sodifrance. Le point de vue que nous adoptons est donc celui du fournisseur de service. Cette pratique tend à se généraliser. Elle permet à des entreprises, dont l'informatique n'est pas le métier premier, de se concentrer sur leur domaine d'origine. Il n'est alors en effet plus de son ressort d'entretenir une armée d'informaticiens avec les contraintes qui s'y attachent (gestion humaine, formations, etc.). Bien sûr, l'externalisation de la maintenance du système d'information, outil structurant pour l'entreprise, ne va pas sans poser de sérieuses contraintes. La qualité du service rendu, la maîtrise des coûts et des délais sont autant d'éléments contractuels qui impliquent que le processus de maintenance soit bien maîtrisé et instrumenté. Parmi les services rendus par la société réalisant la maintenance, on distingue deux types d'intervention: les corrections d'incidents et les demandes de maintenance.

Deux sources différentes d'incidents ont été identifiées. Il peut être observé par un utilisateur lors d'une transaction avec le système d'information. Il peut être détecté par l'étude des listings traçant les passages des chaînes de traitement batch durant la nuit. Dans ces deux cas le délai de correction est fixé contractuellement. Il est de 4 heures pour un incident batch, et de 8 heures pour un incident transactionnel.

Les demandes de maintenances sont de trois types différents. Les demandes de maintenance corrective correspondent à des dysfonctionnements observés sur certains composants du système. Elles peuvent donc être assimilées à des corrections d'incident. Elles s'en distinguent toutefois par la nature du problème. Dans le cas d'une demande de maintenance corrective le dysfonctionnement reporté ne nuit pas à la bonne marche du système. C'est-à-dire qu'il n'est pas bloquant, et que l'utilisateur parvient à se satisfaire du niveau de service rendu. Les demandes de maintenance évolutive consistent à étendre les

fonctionnalités d'un composant du système. Dans certains cas extrêmes ces demandes de maintenance évolutive peuvent donner lieu à des mini-projets, impliquant plusieurs développeurs sur une durée conséquente. Enfin, les demandes de maintenance réglementaire répondent à des changements dans la réglementation.

Les processus mis en œuvre pour répondre à un incident ou à une demande de maintenance comportent deux volets. Il y a bien sûr le processus technique de correction, assimilable à un processus de développement logiciel du type analyse, conception, implémentation, tests unitaires, tests d'intégration. Il y a également le processus de communication avec le client: réception de la demande, réunions de confrontation, négociations du devis, livraison du produit fini. Ces deux volets sont tellement imbriqués que nous n'avons pas cherché à les distinguer de façon explicite.

La plate-forme de maintenance qui a fait l'objet de notre expérimentation est distribuée sur deux sites, Orléans et Paris. Les infrastructures matérielles et logicielles sont totalement fournies par le client, mis à part un certain nombre d'outils spécifiques au processus de TMA, tel que Essor, l'outil de cartographie et de rétro-documentation développé par Sodifrance. Le système d'information de la société cliente est divisé en trois parties:

- La partie production, qui contient les composants logiciels qui sont effectivement utilisés dans les traitements quotidiens par les utilisateurs.
- La partie recette, qui est une copie de la partie production, et qui permet de tester et de valider les composants livrés par l'équipe de maintenance avant leur mise en production.
- La partie étude, qui est l'environnement de travail des collaborateurs de la plate-forme de maintenance. C'est une copie partielle de l'environnement de production permettant la modification de composants existants et le développement de nouveaux composants.

Les environnements de production et de recette sont sous la responsabilité unique du client. Les équipes de la plate-forme ne peuvent y avoir accès directement. L'environnement d'étude est sous la responsabilité de l'équipe de maintenance.

7.3. Le processus modélisé

Le processus de TMA est divisé en quatre phases principales (Figure 97). Il y a d'abord la phase d'initialisation où l'on va extraire les informations véhiculées par la demande de maintenance. Il y a ensuite la phase de pré-instruction ou une réunion peut être organisée avec le client afin de clarifier certains points de la demande. Si la demande concerne une maintenance, un devis est établi. S'il s'agit d'une correction d'incident ou lorsque le devis de la maintenance a été accepté par le client, l'intervention est effectuée. Des retours sont possibles à partir de la phase de réalisation vers les phases de pré-instruction et de devis. Cela arrive lorsque l'on constate que la demande du client est mal formulée, c'est-à-dire que

les besoins exprimés ne correspondent pas aux besoins réels. Il y a alors délivrance d'un REV (Retour sur évolution) par le client. Comme nous le verrons, cette divergence n'est détectée qu'en toute fin de chaîne. Enfin, nous ne l'avons pas indiqué pour ne pas surcharger le schéma, une intervention peut à tout moment être annulée, suspendue ou lotie (divisée en plusieurs sous-interventions).

Il y a deux événements déclencheurs d'un processus d'intervention. Le premier est la réception d'un mail du client contenant une fiche de demande d'intervention. Le second est limité aux incidents batch. Le processus de correction est démarré à la détection d'une erreur dans les listings résultant des passages des chaînes de traitement. Ce travail d'analyse est réalisé par une personne de l'équipe de TMA tous les matins.

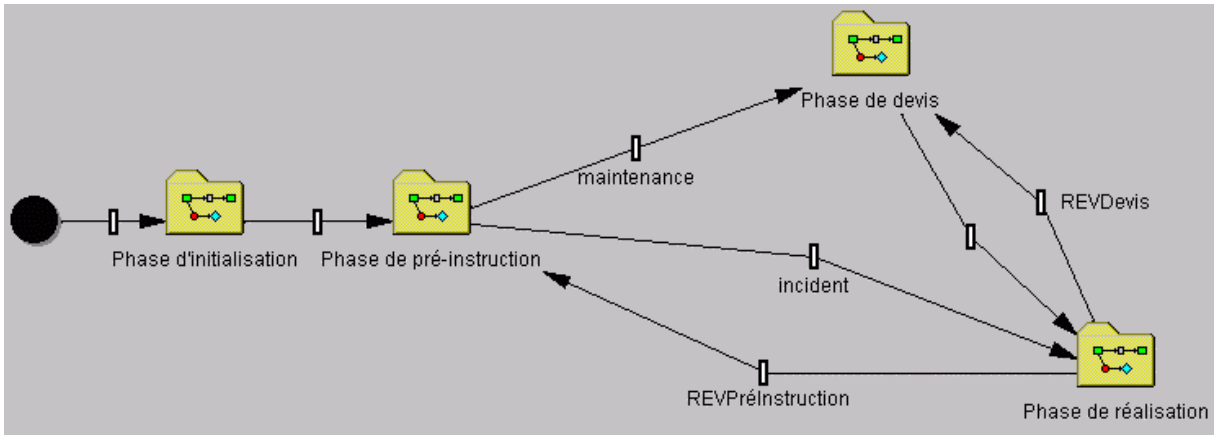


Figure 97. Les phases d'une TMA

La phase d'initialisation (Figure 98) consiste à enchaîner deux traitements automatiques. Le premier récupère un certain nombre d'information à partir du numéro de l'intervention. Un système de numérotation des interventions a été défini en collaboration avec le client afin de faciliter les réunions de confrontation. Chaque numéro véhicule un certain nombre d'informations selon un code bien précis. A partir du seul numéro, on va ainsi pouvoir extraire le domaine et le secteur fonctionnel concernés par l'intervention, le site (Paris ou Orléans) à l'initiative de la demande, le fait que l'intervention porte sur un incident ou une maintenance, et si c'est un incident, le fait qu'il soit transactionnel ou batch. Dans un deuxième temps la fiche de demande va être analysée (sauf dans le cas des incidents batch où aucune fiche n'est émise par le client). Ainsi, toutes les informations qu'elle contient (auteur, expression du besoin, priorité, date de correction souhaitée, etc.) vont en être extraites et être intégrées dans le workflow de façon automatique, sans aucune intervention humaine. Enfin, la dernière étape de cette phase consiste au contrôle de ces données par le responsable de suivi. Quatre rôles différents ont été identifiés dans ce processus:

- Le responsable de suivi, qui est le responsable de l'intervention vis-à-vis du client et qui décide de l'affectation des ressources
- Le responsable de traitement, qui rédige le devis et dirige la réalisation
- Les réalisateurs, qui interviennent juste pour l'implémentation
- Le responsable de la TMA, qui doit donner son visa à tous les devis présentés au client.

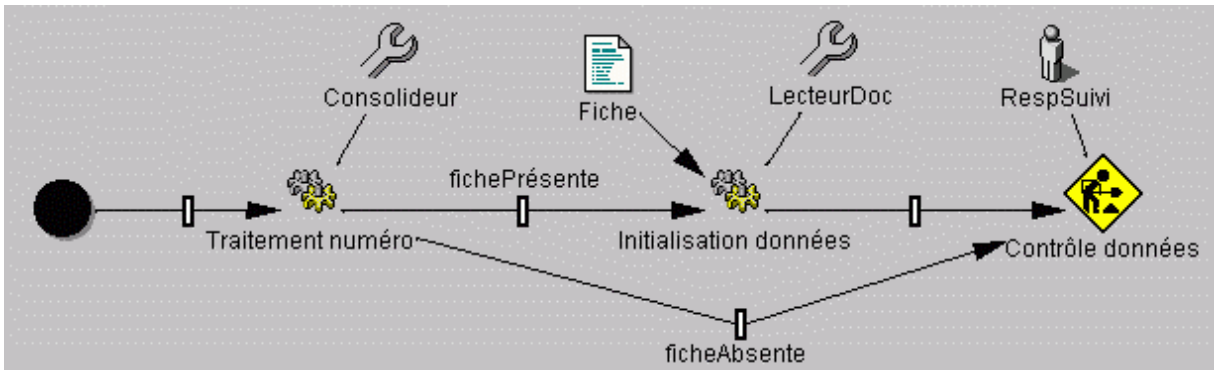


Figure 98. La phase d'initialisation

La phase de pré-instruction (Figure 99) permet au responsable de suivi de proposer une réunion au client si le contenu de la demande est imprécis. Nous n'avons pas fait apparaître le client dans ce processus. Toutes les interactions entre l'équipe TMA et le client se font par envoi de mail. C'est le cas ici où le responsable du suivi peut envoyer une demande de réunion par mail au client. Dans le cas où une réunion a été demandée, le processus reste en suspens jusqu'à sa date d'organisation.

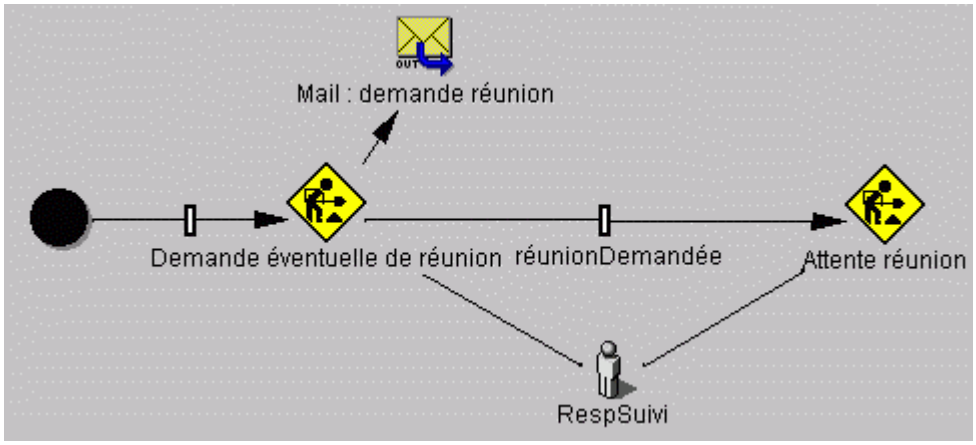


Figure 99. La phase de pré-instruction

La phase de devis (Figure 100) n'est exécutée que dans le cas des demandes de maintenance. En effet, contrairement aux incidents dont le délai de correction est fixé de façon contractuelle, les délais et les ressources associés à la réalisation des évolutions sont négociées au cas par cas. Cette négociation se fait par l'intermédiaire d'un devis produit par la plate-forme de TMA. Ce devis peut être discuté jusqu'à ce qu'il soit définitivement accepté ou refusé par le client.

Le responsable de suivi commence par sélectionner quelqu'un de son équipe pour jouer le rôle de responsable de traitement. C'est cette personne qui est chargée de rédiger le devis. Une fois que la sélection a été faite, une première version du devis est générée à partir des informations extraites de la fiche de demande. Cette version initiale est ensuite enrichie à la rédaction du devis. C'est dans cette phase que le rédacteur analyse le problème, propose une solution pour le résoudre, et détermine les charges et les délais. Dans sa tâche il peut utiliser l'outil de cartographie Essor, afin de déterminer les composants potentiellement impactés. A partir du nombre et du volume de ces composants, il peut utiliser la base de cotation pour estimer les charges. La base de cotation est un tableau

Excel contenant des heuristiques permettant de prévoir des charges pour la modification de composants en fonction de leur complexité (le plus souvent déterminées par le nombre de lignes des programmes). Cette tâche peut être interrompue pour poser au client une question sur un point bloquant. Le processus est alors suspendu jusqu'à réception d'une réponse du client.

Une fois que le devis a été rédigé, il passe par deux niveaux de validation. Tout d'abord, en interne, le responsable de la TMA doit apposer son visa avant qu'il puisse être présenté au client. Puis, le client doit marquer son accord. Le responsable du traitement envoie alors le devis par mail au client. La réponse est reçue par le responsable du suivi, interface entre le client et la plate-forme. Si le devis n'est pas validé, on peut soit recommencer la rédaction, soit annuler purement et simplement l'intervention (cette annulation ne peut être que du fait du client).

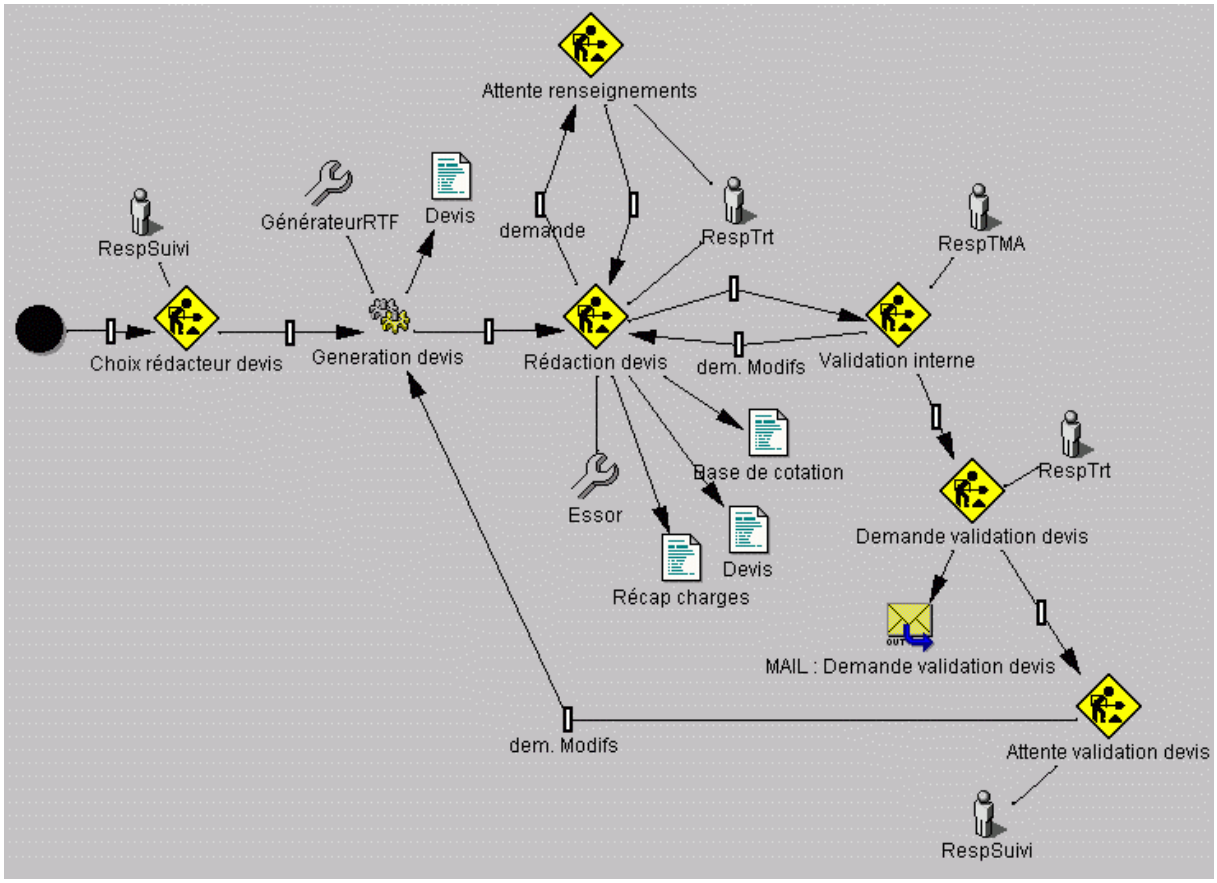


Figure 100. La phase de devis

La dernière phase est la phase de réalisation (Figure 101). Le responsable du suivi commence par sélectionner un responsable du traitement (s'il n'en a pas déjà choisi un) et les réalisateurs qui effectueront l'intervention. Une première version du dossier de recette est ensuite automatiquement générée. Elle est ensuite enrichie tout au long de la réalisation. Le responsable du traitement réalise alors une pré-réalisation. Il découpe le travail à effectuer entre les différents réalisateurs. Puis, les réalisateurs sélectionnés implémentent la correction ou l'évolution, et effectuent les tests unitaires sur les composants dont ils sont

responsables. Les réalisateurs pouvant être jusqu'à cinq, plusieurs instances de la tâche de réalisation peuvent se dérouler simultanément.

Une fois que les réalisations sont terminées le responsable du suivi les récupère et effectue les tests de non-régression et d'intégration. Il complète également le dossier de recette. A la fin de cette tâche il peut soit décider d'un passage en assurance, soit d'une validation par l'expert fonctionnel. Cette décision est prise en fonction d'un certain nombre de critères (urgence de la résolution du problème, complexité, etc.). Le passage en assurance est une validation des composants par l'application d'un certain nombre de traitements dans l'environnement de recette. Il est donc réalisé par les équipes du client. Les scénarios à jouer sont précisés dans le dossier de recette. Si le passage en assurance réussit, la validation de l'expert fonctionnel est demandée. Si l'un des scénarios échoue, la réalisation est à refaire.

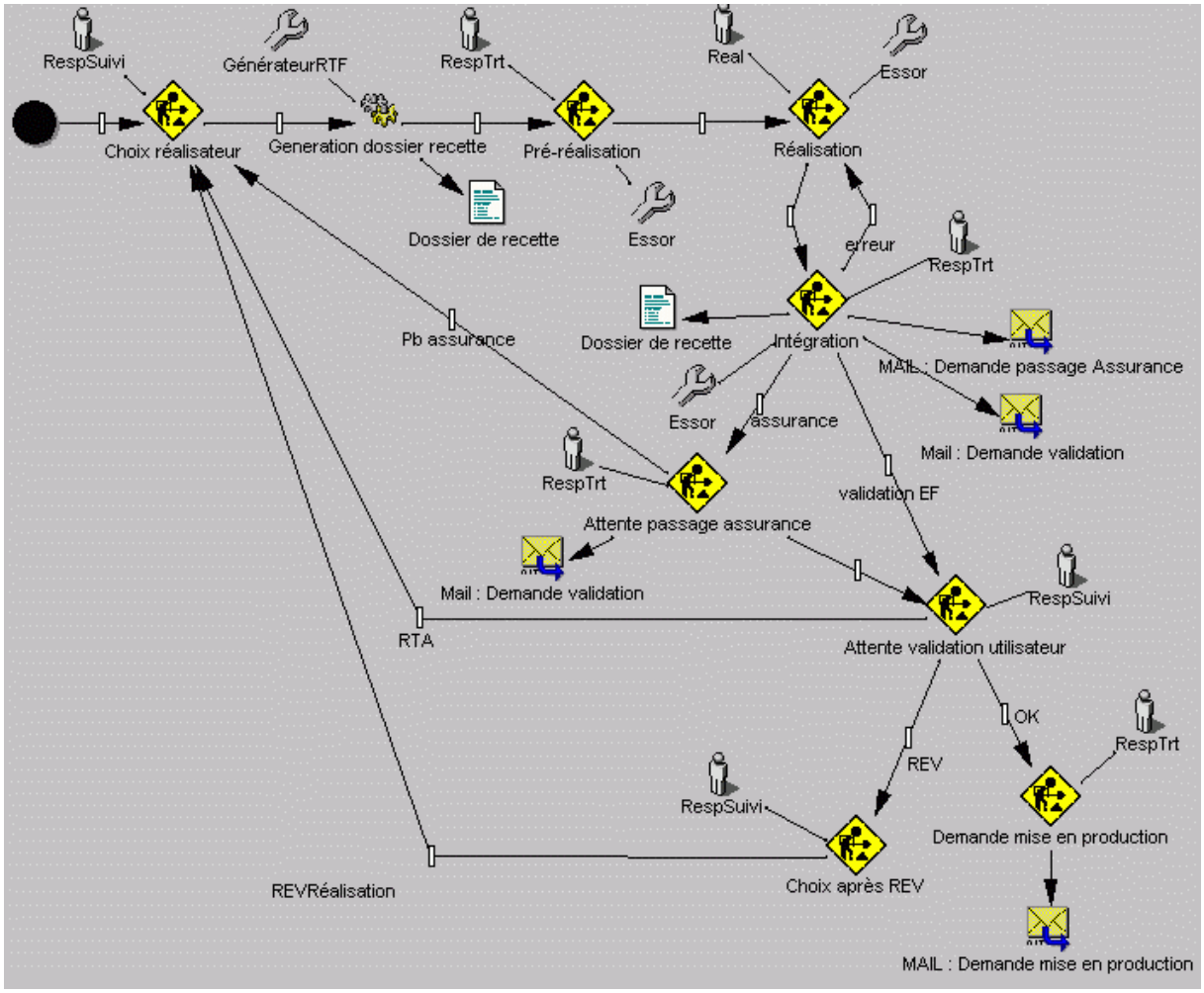


Figure 101. La phase de réalisation

L'expert fonctionnel est une personne du côté client, souvent celle qui est à l'origine de la demande. Il valide donc les aspects fonctionnels, c'est-à-dire qu'il vérifie si le résultat de l'intervention est conforme à ses attentes. Si ce n'est pas le cas, deux situations peuvent se présenter. Des erreurs subsistent dans le composant livré. Un RTA (Retour Technique sur Anomalie) est alors émis et la réalisation doit être recommencée. L'autre cas concerne une

mauvaise expression ou une mauvaise appréciation de la requête initiale. Le composant ne comporte pas d'erreur, mais il ne répond pas aux besoins exprimés. Un REV (Retour sur évolution) est alors émis. Selon la déviation entre la demande initiale et le résultat de l'implémentation, on peut soit recommencer la réalisation, soit retourner en phase de rédaction du devis (si l'analyse ou la conception est à refaire), voire même en phase de pré-instruction s'il est nécessaire de redéfinir tout le cadre de l'intervention. Cette détection tardive dans la détection des déviations entre l'expression des besoins et sa compréhension n'est pas forcément un handicap majeur. En effet, la plupart des interventions portent sur des problèmes très limités, avec des délais de mise en production très courts. De plus, de nombreux contacts informels (et donc absents du modèle) ont lieu entre les équipes de la plate-forme de TMA et celles du client. Enfin, lorsque l'expert fonctionnel donne son aval, les composants modifiés sont livrés au client pour être mis en production.

Dans ce modèle nous avons donc présenté la décomposition du processus en activités et tâches, leur ordonnancement, ainsi que les rôles, les documents et les outils. Les données n'apparaissent pas à l'écran. Toutefois, elles sont très nombreuses. On peut citer par exemple le numéro de l'intervention, la date de la réunion de pré-instruction, la date de mise en production prévue, l'analyse du problème, la solution proposée, etc. De plus, nous avons introduit un nouveau type de donnée (via les *tagged values*). Ce sont les données de type décision. Ainsi, à la fin de la tâche d'intégration de la phase de réalisation, deux décisions peuvent être prises: le passage en assurance ou la demande directe de validation par l'expert fonctionnel. Cette tâche va donc pouvoir mettre à jour deux données décisionnelles, qui s'excluent l'une l'autre. En fait, trois décisions supplémentaires peuvent également être prises lors de cette tâche, tout comme lors de l'ensemble des tâches: lotir, annuler ou suspendre.

7.4. La démarche

Notre objectif dans ce projet était donc d'automatiser le processus de TMA au moyen du moteur de workflow FlowMind. Pour cela nous avons utilisé deux des composants de la chaîne d'ingénierie de processus:

- le modeleur de processus pour la définition du processus de TMA
- Scriptor-G pour la génération du workflow et autres produits dérivés du processus.

Notre travail a donc consisté à définir le processus en collaboration avec quelques personnes de la plate-forme de TMA, ainsi qu'à définir les services de génération permettant d'alimenter le moteur de workflow à partir du modèle de processus. Ces deux activités étaient réalisées en parallèle et de façon incrémentale. Le modèle du processus est un modèle métier, indépendant de toute plate-forme. Les services de génération spécifient la façon dont les concepts métiers sont intégrés dans l'architecture d'implémentation. Ce processus peut être assimilé à un cycle en Y, avec d'un côté un PIM contenant le modèle du processus indépendamment des contraintes dues à la plate-forme d'exécution, et de l'autre

les services permettant leur intégration sur cette dernière (constituée dans notre cas par FlowMind, le serveur HTTP Apache, le serveur de servlet Apache Jserv, la base de données MySQL, et l'ensemble des applications déployées dans le cadre de la TMA).

Des informations nécessaires au pilotage des services de génération ont toutefois été définies au sein du modèle métier par l'intermédiaire de *tagged values*. On peut donc considérer qu'on a deux modèles de processus (même si, physiquement, il n'y a qu'un seul modèle). Le premier modèle, sur lequel ont travaillé les experts métier, est une description générique du processus. Le second modèle est une adaptation du premier. Des *tagged values* ont été ajoutées afin de décliner le modèle générique pour une plate-forme particulière. Ce travail est donc du ressort de l'intégrateur.

Le premier modèle est ainsi basé sur le méta-modèle tel qu'il a été défini, alors que le second se base sur une extension de ce méta-modèle (en quelque sorte un profil). Cette démarche entre donc totalement dans le cadre de l'architecture du MDA de l'OMG.

Le processus a été modélisé de façon incrémentale. Nous avons choisi de modéliser d'abord les activités, les tâches et leur ordonnancement, c'est-à-dire les transitions et leurs conditions. Nous avons donc dû également définir les décisions qui pouvaient être prises ainsi que les données pertinentes (au sens où l'entend le WfMC, c'est-à-dire les données qui vont permettre d'orienter le flux de contrôle). Une fois que le graphe d'activité du processus a été validé, nous nous sommes attachés à définir le jeu de rôles, puis les documents et les outils. Les données ont été ajoutées à chaque étape, selon les besoins.

La séparation entre ces étapes n'a bien sûr pas été aussi claire dans la réalité que ce que nous avons pu indiquer précédemment. En effet, lorsqu'un expert métier modélise une tâche, on ne souhaite en aucun cas le restreindre à ne saisir qu'une partie de ses connaissances. Même si l'objectif de l'incrément est la définition des tâches, il peut vouloir signaler l'utilisation d'un outil lors d'une tâche. Toutefois, étant donné la pauvreté du modèle à ce moment, il est difficile pour l'intégrateur de paramétrer l'invocation de cette application. Un des avantages de notre solution, par rapport à la plupart des outils de conception de workflow, est qu'un modèle même incomplet peut être exécuté. En effet, lors de la génération du prototype, seules seront retenues les parties du modèle que l'on souhaite valider lors d'un incrément. Ainsi, lors des incréments initiaux, le workflow généré ne prendra en compte que le graphe d'activité. On peut ainsi masquer et démasquer certains aspects de modèle lors de la génération du workflow.

Dans le cas où cet aspect est nécessaire, des choix arbitraires peuvent être faits pour que le workflow puisse tout de même être validé. C'est par exemple le cas des tâches automatiques. Une tâche automatique est une invocation batch d'une application. Si l'application qu'elle encapsule n'est pas décrite, le moteur de workflow générera une exception. Pour pallier ce problème, nous avons utilisé un rôle virtuel à qui toutes les tâches automatiques étaient affectées sous la forme de tâches manuelles. Elles étaient bien décrites comme des tâches automatiques dans le modèle de processus, mais produisaient à la génération des tâches manuelles. Lorsque les outils ont été définis et interfacés correctement, ce *pattern* a été abandonné. Ainsi, au fur et à mesure du projet les services

de génération ont évolué. A tout moment, la version du processus défini avec les experts métier peut-elle être générée et validée. Il n'y a pas forcément besoin d'attendre que le modèle ait atteint un niveau important de complétude et de précision.

L'implémentation du processus automatisé a donc été découpée en deux activités parallèles et itératives : la modélisation de processus et le développement des services de génération (Figure 102). L'application d'une version des services de générations sur une version du modèle permet de produire un prototype, plus ou moins proche du résultat final. Cela permet aux experts métier de valider non pas un modèle statique et générique, mais un prototype dynamique et adapté à leur environnement d'exécution. Les dysfonctionnements identifiés doivent ensuite être analysés pour déterminer s'ils proviennent du modèle ou de sa transformation. Ainsi, si la validation du processus par les experts métier est facilitée, l'analyse des dysfonctionnements nécessite un certain degré de maîtrise de la solution globale pour être en mesure d'en définir l'origine.

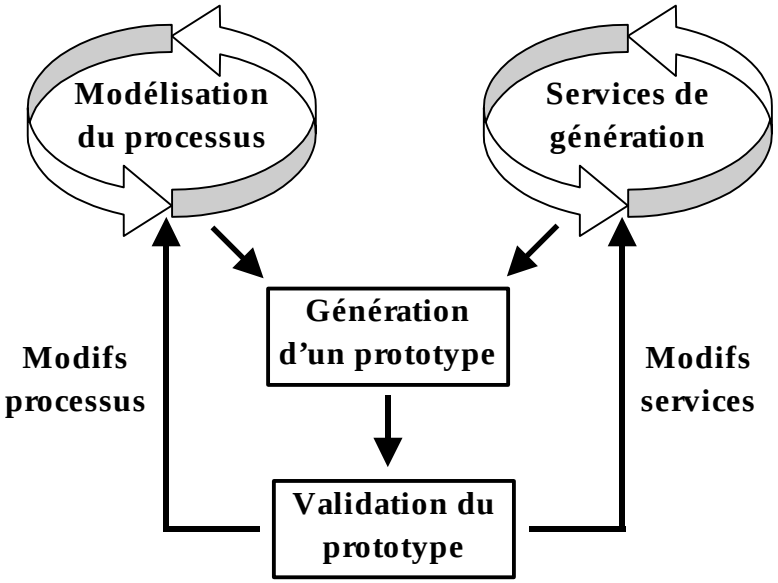


Figure 102. La démarche itérative

7.5. FlowMind

FlowMind est un environnement de gestion de processus. Il est un des composants majeurs de la plate-forme logicielle de la TMA, puisque c'est lui qui supporte l'exécution du processus. Il est édité et distribué par la société Akazi. Cette start-up a été fondée en 1998 et est basée à Meylan (près de Grenoble).

FlowMind se décompose en quatre composants principaux (Figure 103). Le Designer permet de concevoir des modèles de workflow. Le Processor est le moteur d'exécution. Le Reminder est l'interface standard par laquelle l'utilisateur pourra interagir avec le workflow. Enfin, l'InsightManager permet de suivre, de contrôler et d'administrer l'évolution des

différentes instances de processus. Les interconnexions avec le système d'information se font grâce à des composants Java: les FlowBeans.

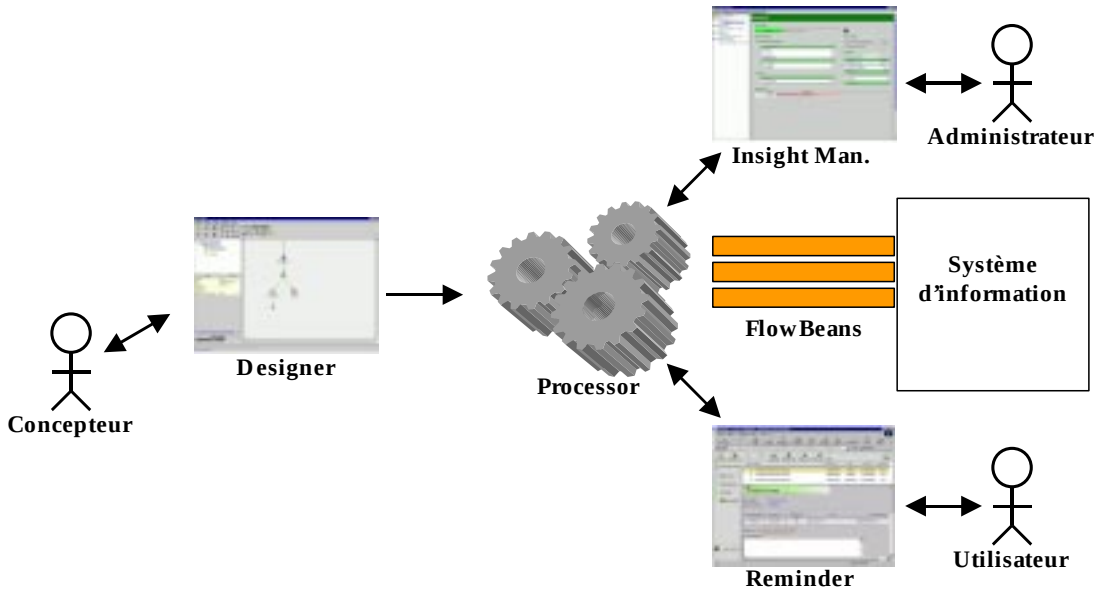


Figure 103. Les différents composants de FlowMind

Le Designer (Figure 104) est un outil graphique de définition de workflow. Il permet de définir la décomposition du processus en étapes, l'enchaînement de ces étapes, les flux de données ainsi que les connexions avec le système d'information.

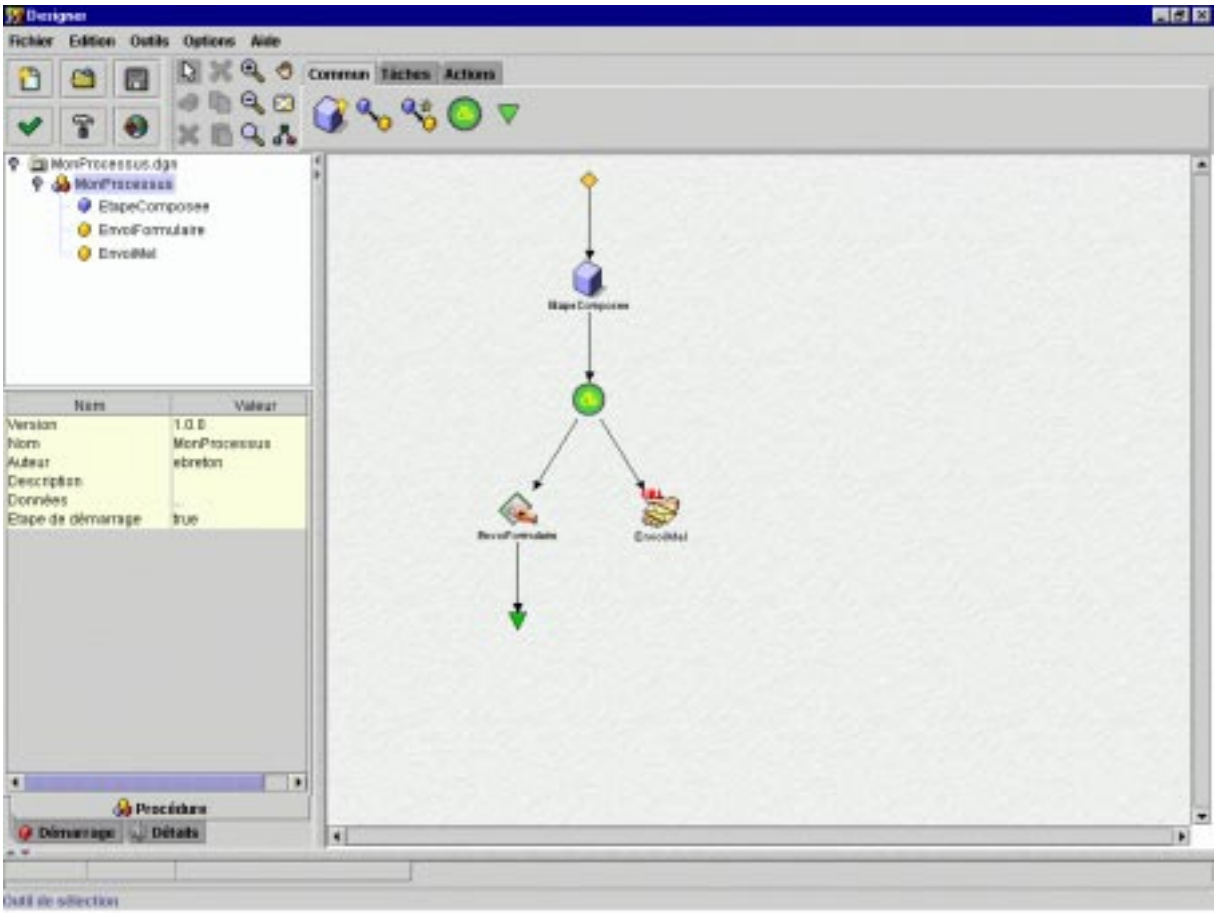


Figure 104. Le module *Designer* de FlowMind

Le méta-modèle sous-jacent des workflows conçus à l'aide du *Designer* est présenté Figure 105. Un processus est constitué d'une étape principale. Cette étape peut être soit une étape simple, soit une étape composée. Une étape composée définit à la fois des étapes de plus bas niveau et un graphe d'activité. Ce graphe est une succession d'invocation d'étapes, soit directement contenue dans l'activité composée, soit dans l'une des activités contenant cette activité composée. L'ordonnancement des étapes est spécifié à l'aide de transitions spécifiant des conditions sur les données définies dans le cadre de l'activité englobante.

Une étape atomique encapsule un FlowBean. Un FlowBean est un composant Java qui implémente un certain nombre de méthodes (*onLoad*, *onStart*, *onRun*) invoquées à divers moment de son cycle de vie. C'est le FlowBean qui permet les communications avec le système d'information. Akazi possède un catalogue de FlowBeans standards, permettant de se connecter à des serveurs de mails, à des bases de données ou encore à des annuaires via des protocoles normalisés (POP3, SMTP, ODBC, LDAP). De plus, il est très facile d'ajouter des FlowBeans spécifiques à un site, à la seule condition que l'application avec laquelle on souhaite communiquer puisse être accédée à partir d'un composant Java.

Chaque étape du processus définit son propre espace de nommage. Toute invocation donne donc lieu à un renommage en entrée (de l'étape englobante vers l'étape invoquée) ou en sortie (de l'étape invoquée vers l'étape englobante). Un renommage en entrée (resp.

sortie) consiste à affecter une expression formée à partir des données de l'étape englobante (resp. invoquée) à certaines données de l'étape invoquée (resp. englobante).

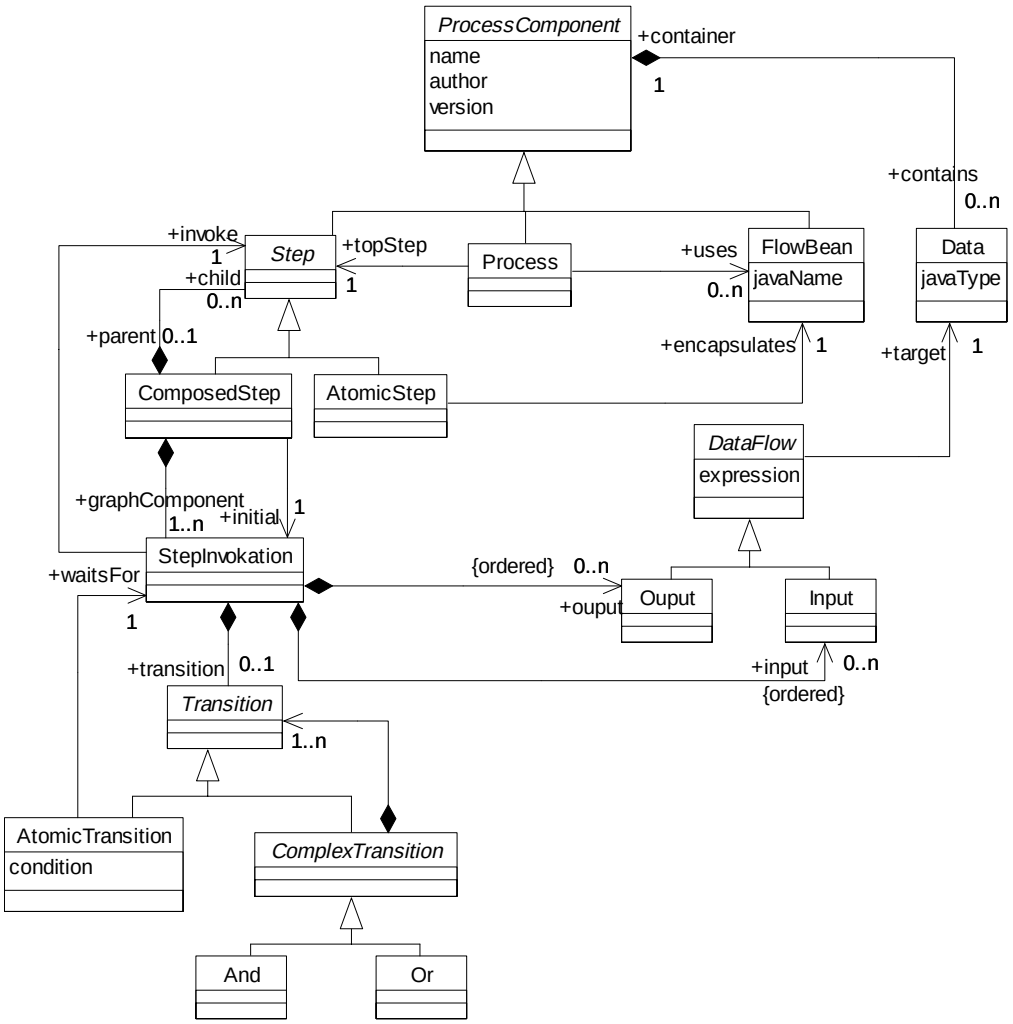


Figure 105. Le méta-modèle de définition de processus FlowMind (version partielle)

Il est intéressant de comparer ce méta-modèle avec celui de notre modeleur de processus. De nombreux concepts présents dans le modeleur de Sodifrance sont absents du Designer. On peut citer par exemple les outils, les rôles ou encore les documents. De plus le fait que chaque étape soit un espace de nommage alourdit considérablement la définition d'un processus. Surtout si l'on tient compte du nombre de renommage que cela implique. Par contre, cette technique permet un niveau de réutilisabilité bien supérieur à celui de notre modeleur. On se trouve donc ici face à un outil défini pour des informaticiens. Il est par contre beaucoup moins bien adapté pour des experts métier ou des organisateurs.

Les modèles de workflow conçus à partir du Designer sont stockés dans un format textuel nommé FMP. C'est ce format que nous utilisons pour générer des workflows FlowMind à partir de notre modeleur.

Une fois qu'un processus a été défini, il peut être chargé dans le Processor pour être exécuté. Les interactions avec les utilisateurs se font par des formulaires HTML. L'envoi d'un formulaire à un utilisateur est spécifié dans le processus par l'invocation d'un FlowBean spécifique. Ces formulaires peuvent être enrichis au moyen d'un langage spécifique (le Tagazi) pour accéder aux données du processus ou obtenir un certain nombre d'informations sur le contexte d'exécution. Ces formulaires peuvent être visualisés au moyen du Reminder (Figure 106), qui permet à un acteur de visualiser et de valider l'ensemble des tâches qui lui sont assignées. Ce Reminder peut être adapté selon les sites, et il peut être intégré au sein d'un portail d'entreprise.

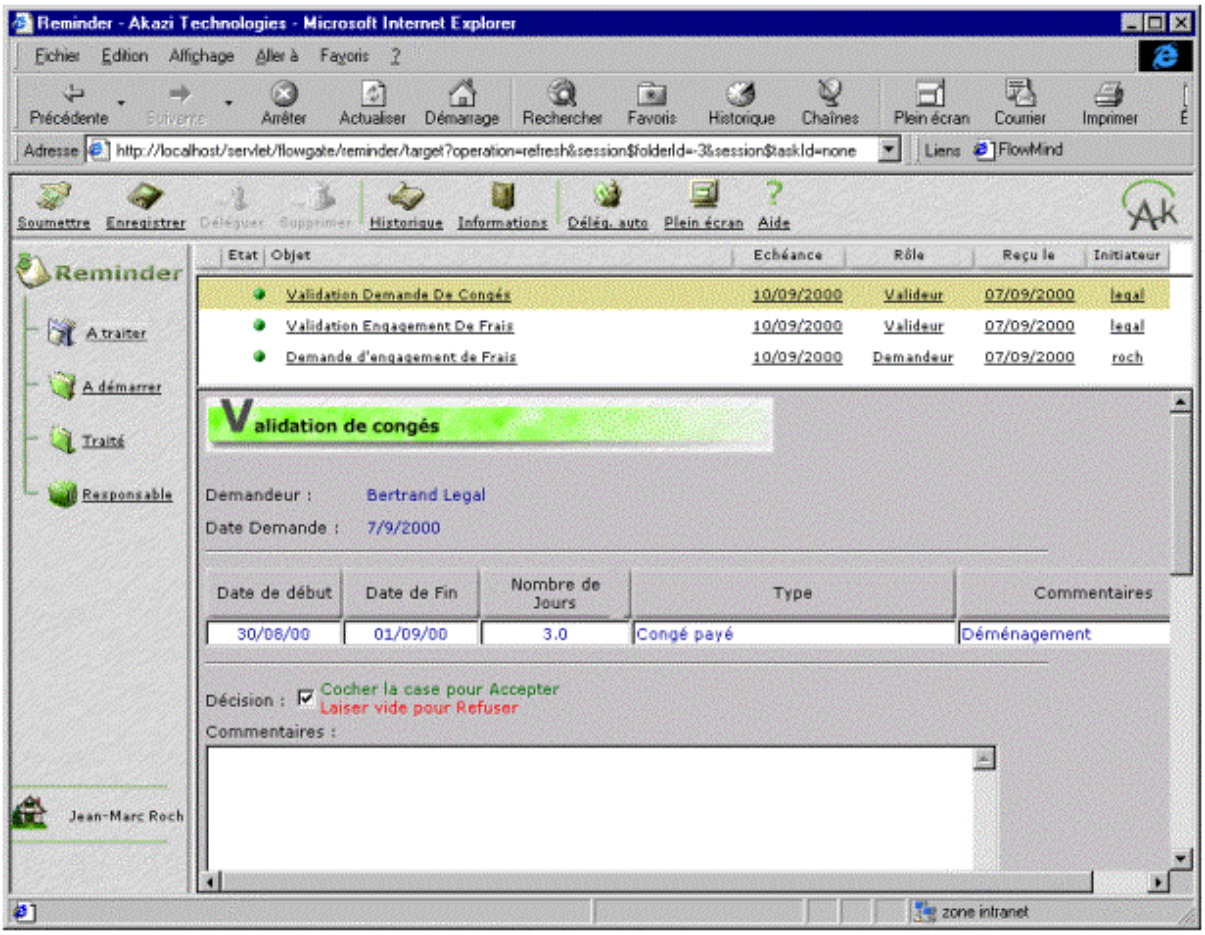


Figure 106. Le Reminder standard (extrait de la documentation FlowMind)

La dernière version de Flowmind a apporté deux innovations importantes. D'une part, le langage FMP a été remplacé par un langage XML. De plus, le Reminder peut aussi intégrer des pages JSP (il ne se limite plus au seul langage propriétaire Tagazi). Enfin, comme nous l'avons déjà mentionné, un partenariat technique a été établi entre Sodifrance et Akazi pour la définition d'un modeleur de processus se situant en amont du Designer.

7.6. Le processus de TMA automatisé

Les informations contenues dans le modèle de processus vont être exploitées pour produire un ensemble d'artefacts. Parmi ces artefacts, on peut citer le modèle de workflow, les interfaces personne machine et la documentation en ligne. Le modèle de workflow est un fichier écrit dans le langage FMP. Il définit les activités et leur ordonnancement, les flux de données, les invocations de FlowBeans. Il est destiné à alimenter le Processor. Les formulaires sont des pages HTML incluant des tags spécifiques (Tagazi). Ils permettent aux collaborateurs de la TMA de connaître les tâches qui leur sont affectées. Ils donnent accès aux documents (Figure 107) et aux applications interactives (Figure 108). Dans le premier exemple, l'application Essor est invoquée depuis le menu du formulaire. Dans l'exemple suivant, la fiche précisant la requête initiale est également accédée à partir du menu. De cette manière, le poste de travail d'un utilisateur est totalement intégré. L'ensemble des outils et documents susceptibles d'être utilisé sont accessibles depuis un même endroit, cette liste étant adaptée aux besoins particuliers à la tâche en cours.

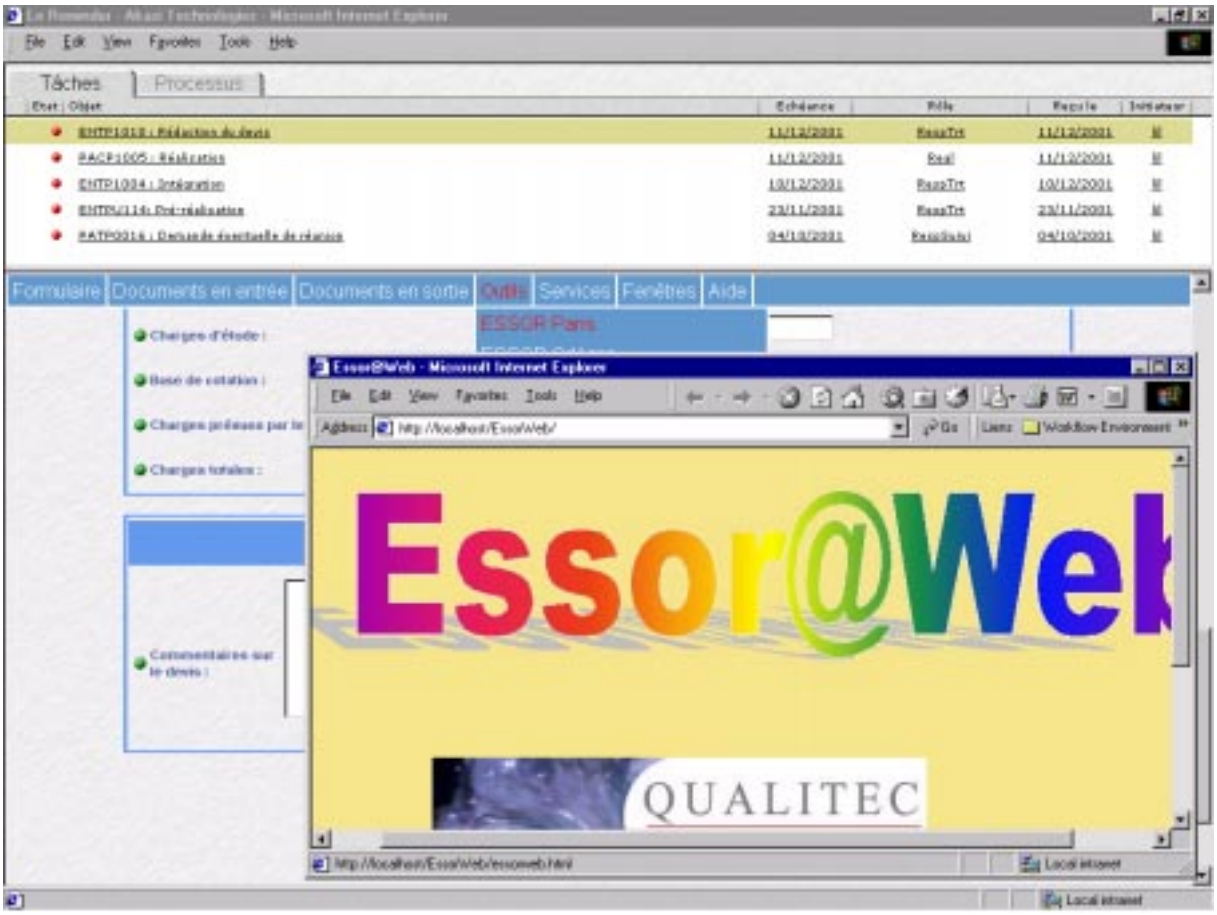


Figure 107. Intégration des applications

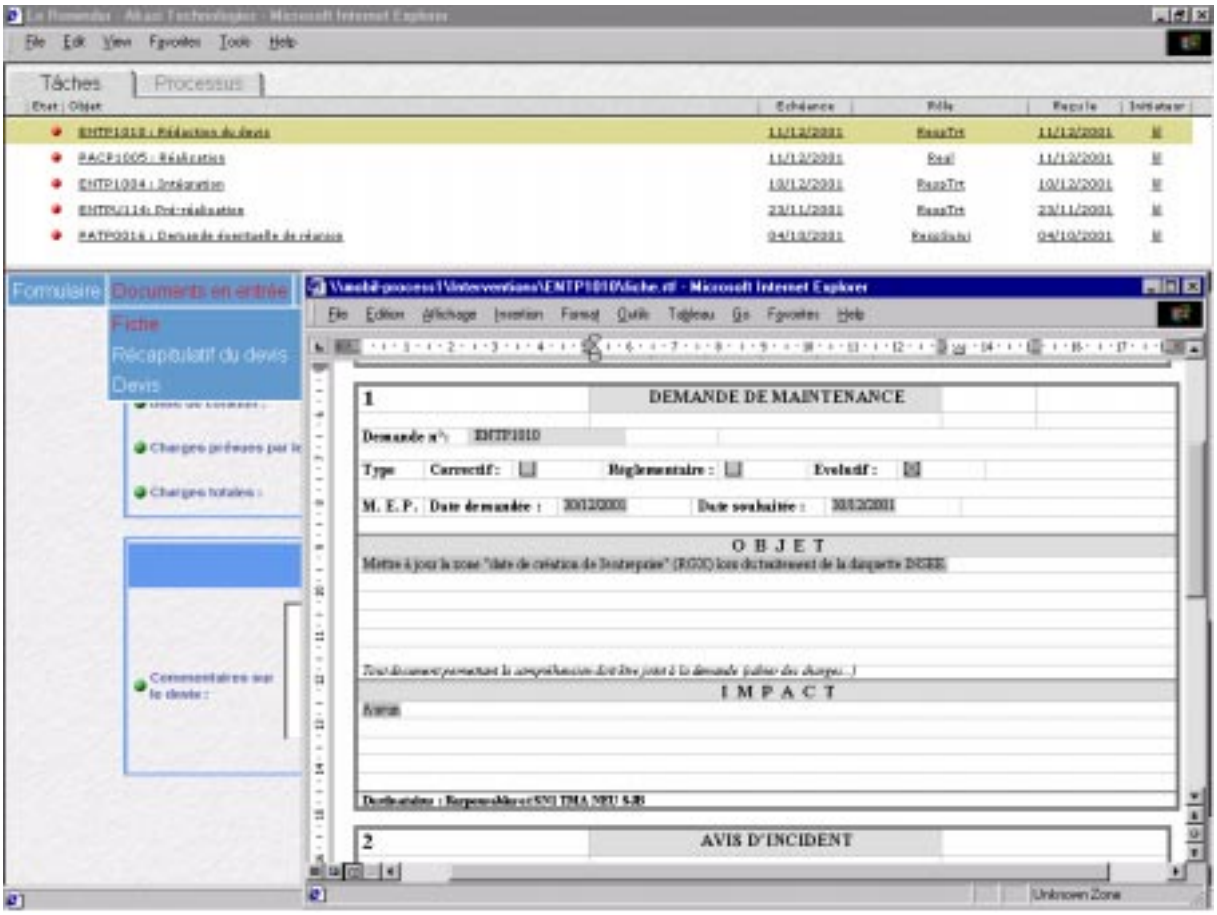


Figure 108. Intégration des documents

Les formulaires constituent également un espace de travail pour l'utilisateur où il peut renseigner des champs et prendre des décisions (Figure 109)

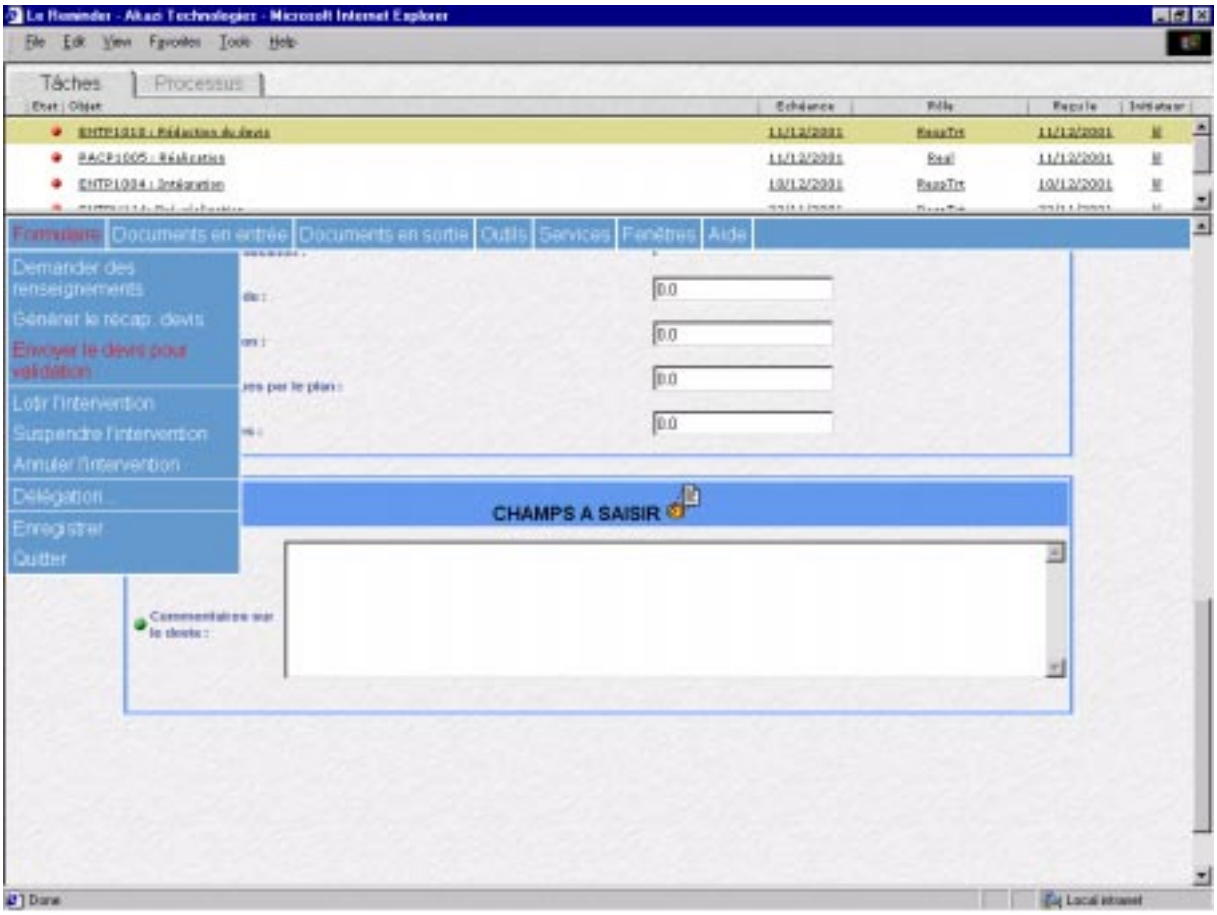


Figure 109. Prise de décision à l'issue d'une tâche

Une des attentes principales vis-à-vis du workflow était de pouvoir disposer d'un suivi à jour et exhaustif. Cette fonctionnalité est possible car le workflow constitue le point de passage obligé pour le traitement des interventions. Beaucoup de moteurs de workflow proposent d'ailleurs des historiques et des statistiques sur les processus effectués. Pour certaines raisons (pérennité de la solution, interrogation facile de la base de données par d'autres applications telles que la facturation, etc) nous avons souhaité effectuer le suivi non pas à travers le support de persistance du moteur de workflow mais via une base de données propriétaire. Cette base de données permet de stocker les données du workflow jugées pertinentes au format désiré par les utilisateurs. Par exemple l'état d'avancement d'une intervention est vu au travers d'un certain nombre d'états qui ne correspondent pas directement à des tâches, mais à des sous-parties du processus.

Le problème auquel nous nous sommes heurtés était la spécification des tâches de mises à jour automatique de la base de données à l'intérieur du workflow. Il était hors de question que ces tâches soient représentées explicitement dans le processus métier car cela l'alourdirait énormément, le rendant moins lisible (comparer Figure 110 et Figure 111). D'autre part cette spécification n'était pas du ressort de l'expert métier et nous ne souhaitions pas déstructurer le modèle de processus fourni.

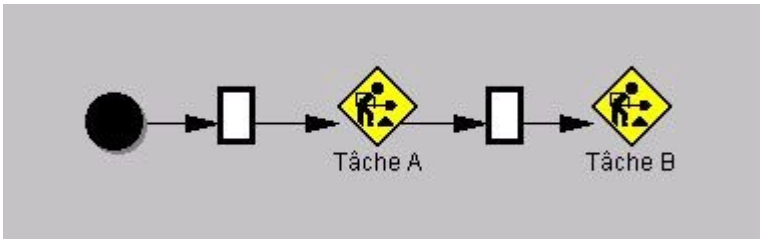


Figure 110. Exemple de processus non surchargé par les mises à jour de la base de suivi

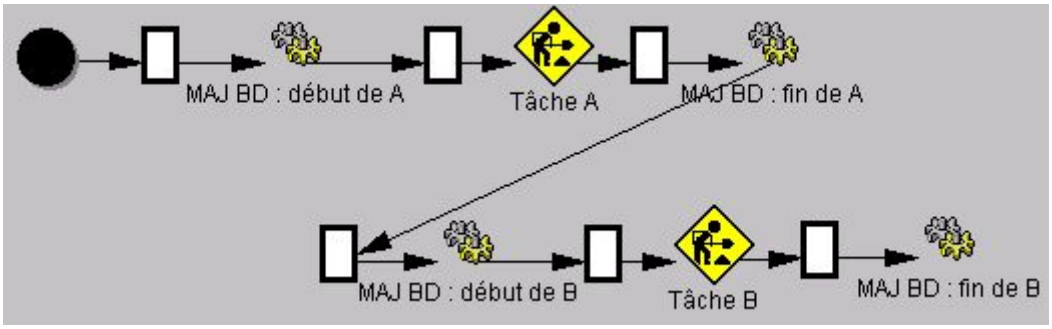


Figure 111. Le même processus surchargé par les mises à jour de la base de suivi

Pour éviter ce problème nous avons donc choisi de générer ces accès à la base de données de suivi. Ainsi la complexité n'est pas au niveau du modèle, qui n'est pas « pollué » par des considérations extérieures au seul processus, mais au niveau des scripts de génération. Pour cela nous avons mis au point une règle de génération spécifique indiquant qu'à chaque étape (activité, tâche et même processus) du modèle ne correspondait pas une étape simple dans le workflow mais une étape composée sur le modèle suivant de la Figure 112.

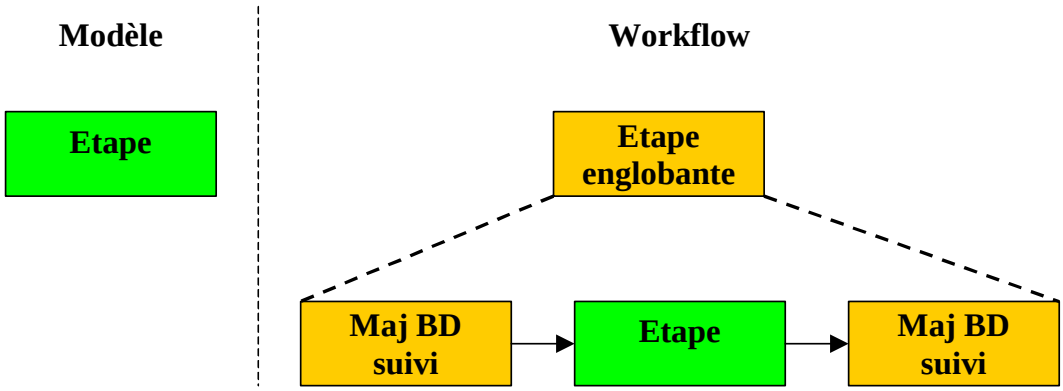


Figure 112. Le pattern de correspondance entre une étape du modèle et une étape du workflow

Ainsi, la base de données de suivi est mise à jour de façon totalement transparente pour l'utilisateur. Une fonctionnalité de consultation en ligne a été développée. Toutes les personnes connectées peuvent avoir une vue de l'avancement de chacune des interventions (voir Figure 113 et Figure 114).

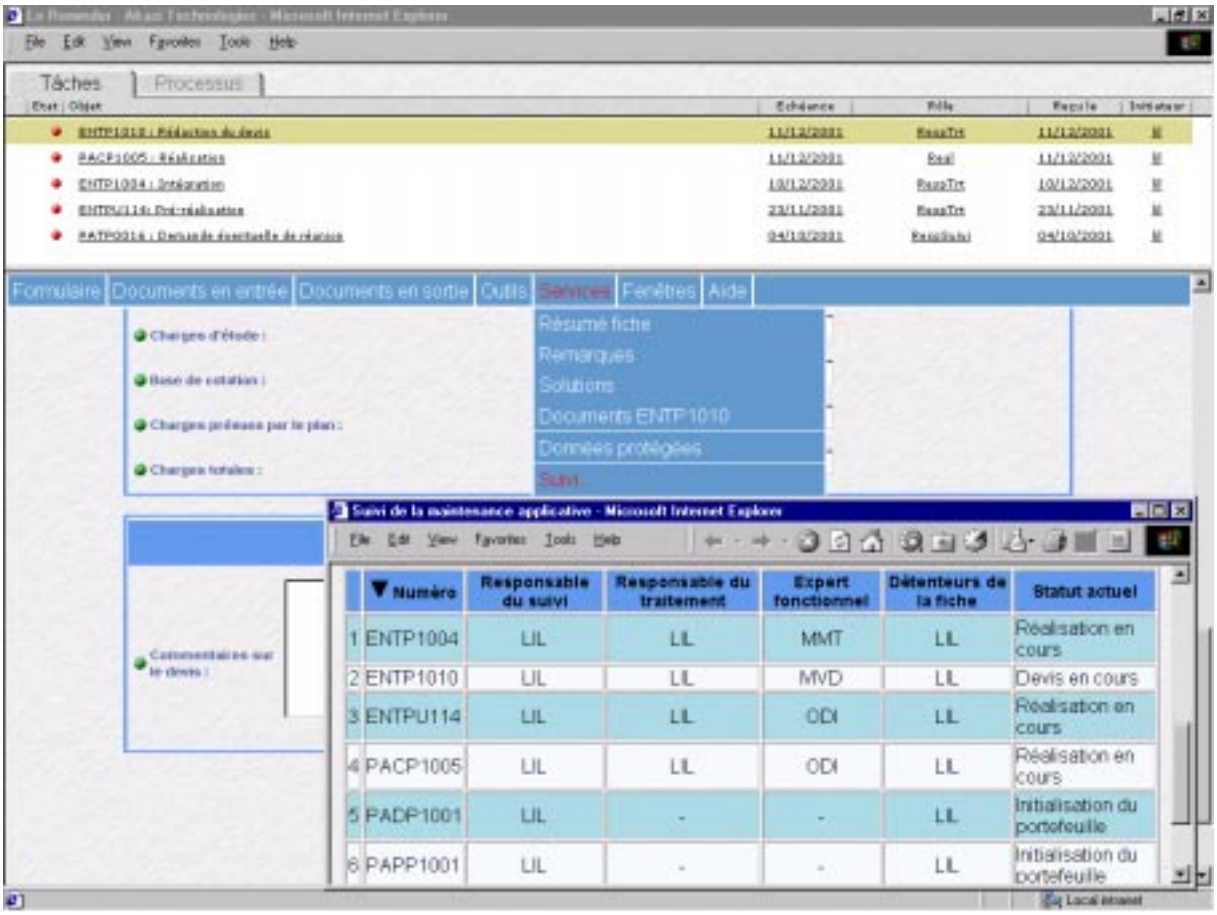


Figure 113. Intégration du suivi en ligne

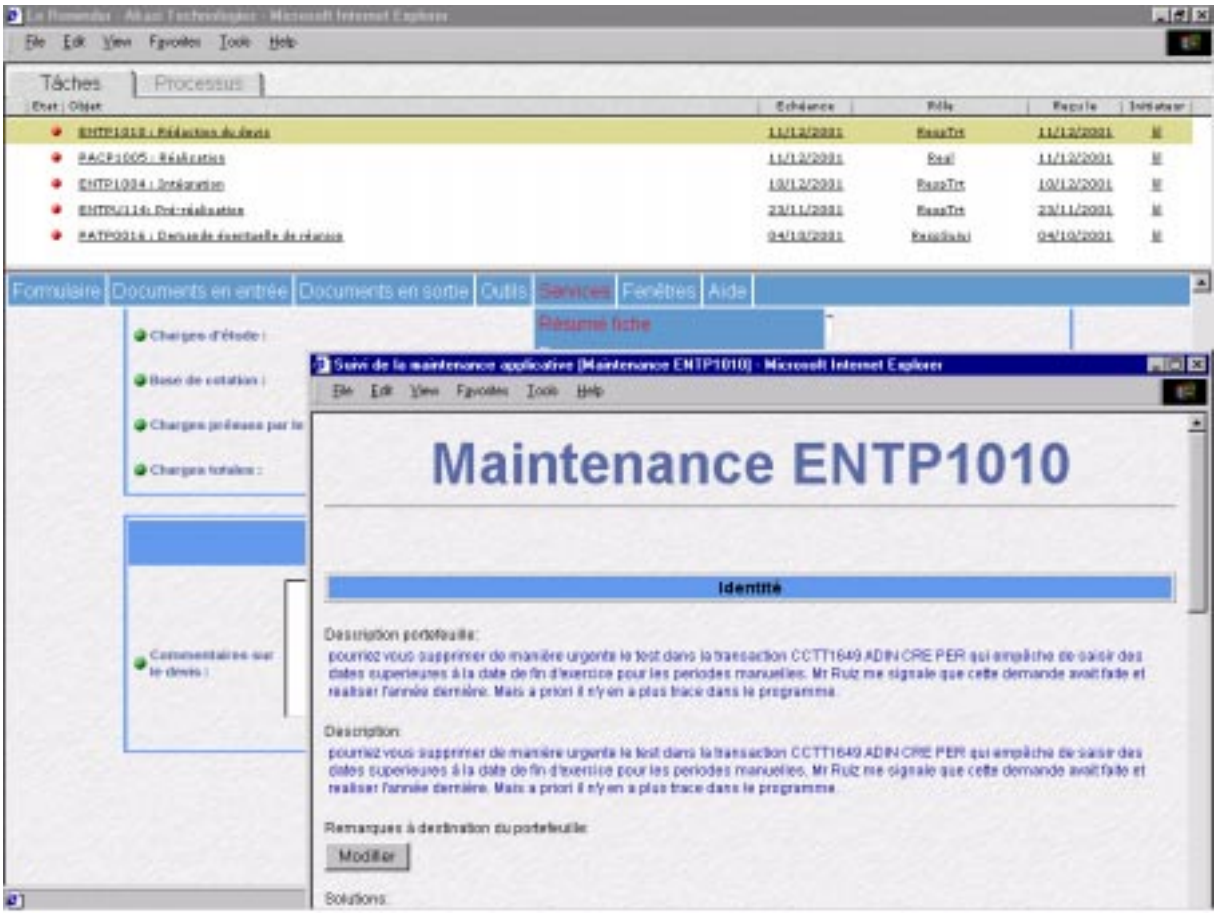


Figure 114. Descriptif d'une intervention en cours

Enfin, un historique technique a également été mis à disposition des réalisateurs. Toutes les informations techniques saisies au cours du processus sont stockées. De la même manière que pour le suivi, la collecte des informations est systématique et transparente pour les utilisateurs. Ainsi, en faisant une recherche sur quelques mots-clés, le réalisateur peut très vite voir si le problème qui lui a été soumis ou un problème du même type, a déjà été traité, qui a réalisé l'intervention, à quelle date, etc. On peut par exemple chercher l'ensemble des interventions qui ont traité la notion de contrat (Figure 115).

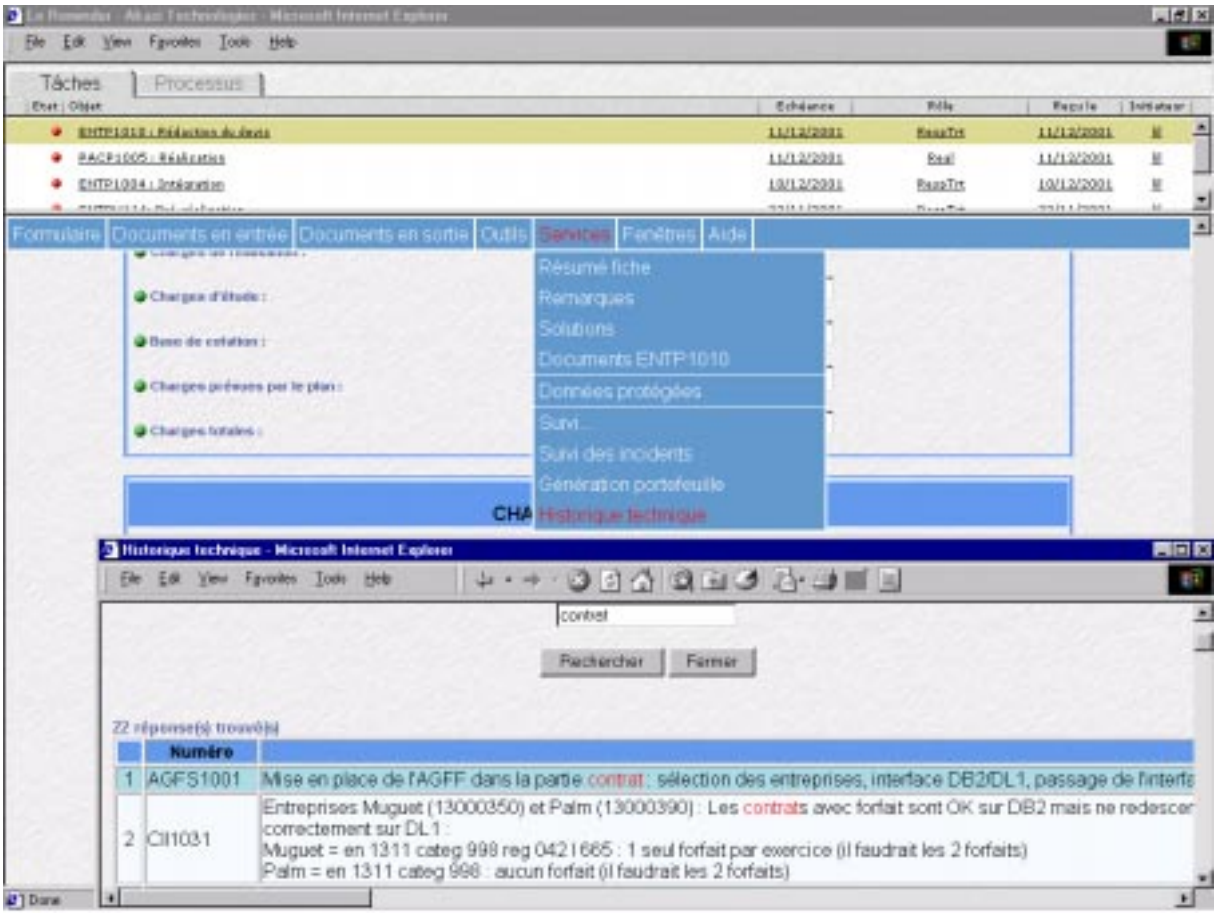


Figure 115. Intégration de l'historique technique

7.7. Conclusion

La mise en production d'une première version du processus de TMA a eu lieu fin 2000. Des versions ultérieures ont été réalisées afin de prendre en compte de nouveaux besoins et de corriger certaines parties du processus. Il est aujourd'hui utilisé par une cinquantaine d'utilisateurs sur les deux sites de Paris et d'Orléans. Cette automatisation du processus constitue une différentiation et une valeur ajoutée certaine pour l'offre de TMA proposée par Sodifrance. Le gain en terme de performance apporté par le déploiement de cette solution a été estimé à dix jours homme par mois. Il concerne principalement l'activité des chefs de projet qui n'ont plus à passer entre une demie-journée et une journée par semaine à collecter des renseignements pour remplir leurs états d'avancement. Ce projet a été sélectionné par Akazi pour faire partie de ses « success stories » [1].

Sur le plan technique, ce projet nous a permis de valider les différents composants que nous avions développés sur un processus industriel de grande ampleur. Tout d'abord les concepts définis dans le méta-modèle ont couvert l'ensemble des besoins d'expression des experts métiers. Les *tagged values* ont permis l'adaptation du modèle générique à la plate-forme d'exécution. La définition du processus à l'aide du modeleur n'a pas non plus posé de problèmes aux experts métier. Les services de génération ont permis de produire des

workflows exécutables tout au long du projet, que ce soit à des fins de validation ou de déploiement. La solution que nous avons développée a donc pu démontrer sa pertinence et son efficacité dans ce cadre qui était, rappelons-le, un des objectifs prioritaires de Sodifrance.

Par contre, nous n'avions jusqu'ici aucunement tenu compte des réactions que pourraient engendrer ces travaux chez les utilisateurs. Ce projet nous a permis de nous confronter à cette réalité du terrain. Jusqu'ici, chaque collaborateur de la plate-forme de TMA affirmait suivre le processus décrit dans les guides méthodologiques. Sa modélisation, et plus encore les démonstrations de prototypes, ont permis de mettre à jour des divergences profondes sur les interprétations qui en étaient faites. La définition d'une version du modèle débouchait sur un premier consensus, consensus qui disparaissait lors des mises en situation. C'est pourquoi la génération immédiate de prototypes à partir de modèles même incomplets nous apparaît comme une solution des plus pertinentes à ce problème. Les éventuels désaccords subsistant après la modélisation sont ainsi découverts beaucoup plus rapidement.

Pour mesurer les gains apportés par notre approche au processus de TMA, on peut se baser sur l'échelle d'évaluation fournie par le CMM. Avant nos travaux, les processus étaient décrits dans divers documents textuels. Un savoir-faire avaient été acquis et un certain nombre de pratiques avaient été identifiées. Le processus de TMA de Sodifrance étaient donc situé au niveau 2 de l'échelle CMM, c'est-à-dire au niveau *Repeatable*. Notre intervention a consisté à permettre la définition et l'adaptation d'un processus standard. De plus, les exécutions de ces processus sont mesurées en terme de charges, de délais, de nombres d'anomalie, etc. Le processus de TMA de Sodifrance tel que nous l'avons instrumenté se situe donc au niveau 4 de l'échelle CMM, le niveau *Managed*. Toutefois, nous n'avons pas mis en place le processus d'utilisation systématique de ces mesures pour une optimisation permanente du processus. Nous ne pouvons donc prétendre au niveau 5 (*Optimizing*) tel qu'il est défini par le CMM.

8. Intégration d'un outil de planification

8.1. Introduction

La réalisation suivante s'est faite dans le cadre de prototypes de processus de développement logiciel. Comme nous l'avons précédemment évoqué, on observe une formalisation croissante des procédés de développement logiciel. On est passé de méthodes détaillées, de type OMT, à des processus précis basés sur des méta-modèles. C'est l'exemple du RUP qui s'est aligné sur SPEM. Ce travail de formalisation répond à des besoins en termes de qualité, de délai et de réduction de coûts. La définition du processus est d'autant plus cruciale que la complexité des systèmes d'information est en augmentation constante (tant au niveau de la taille que des types d'artefacts manipulés) et que les organisations sont réparties sur différents sites.

Une des particularités des processus de développement logiciels, en comparaison, par exemple, avec les processus administratifs, est qu'ils peuvent impliquer un nombre important d'acteurs sur des périodes assez longues. De plus, les durées de ces affectations, ainsi que celle du projet global, ne peuvent qu'être estimées de façon imprécise en début de projet. Cette estimation peut être revue à tout moment en cours de projet.

L'objectif de ce prototype est d'intégrer trois vues différentes sur le processus :

- Celle du responsable méthode, qui définit le processus, c'est-à-dire les différentes étapes, leur ordonnancement, les rôles et les documents échangés,
- Celle du chef de projet, qui planifie le projet, en indiquant des délais et des charges pour les différents réalisateurs,
- Celle des différents réalisateurs, qui remplissent les rôles définis par le responsable méthode en fonction des affectations planifiées par le chef de projet.

Chacun d'entre eux dispose d'un outil différent, adapté à ses besoins, pour interagir avec le processus. Pour ce prototype, les outils mis en œuvre sont:

- Le modeleur de processus pour décrire les processus,
- Le progiciel MS-Project de la société Microsoft pour leur planification,
- Le moteur de workflow FlowMind pour la distribution du travail.

Nous avons vu dans le chapitre précédent une transformation de modèle pour alimenter le moteur de workflow FlowMind à partir de notre modeleur. Pour ce prototype

nous ajoutons une application supplémentaire : MS-Project. La collaboration entre les différents outils se fait de la façon suivante (Figure 116) :

- Le processus est défini par l'intermédiaire du modeleur,
- Il est ensuite exploité pour alimenter l'outil de planification et le moteur de workflow,
- L'outil de planification et le moteur de workflow échangent des informations. La définition des affectations et des échéances est faite avec MS-Project. FlowMind utilise ces informations et met à jour la planification en signalant le démarrage et la fin des tâches.

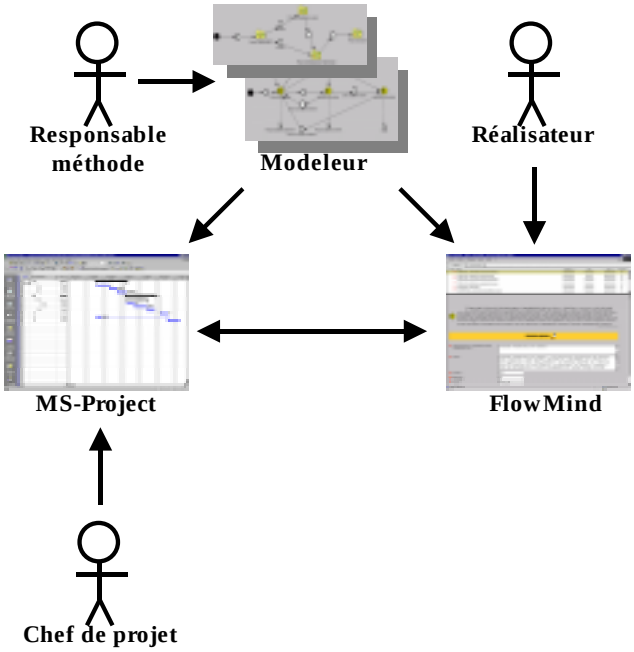


Figure 116. Intégration de la définition, de la planification et de l'exécution

Nous commençons par présenter l'outil de planification MS-Project. Nous étudions ensuite la transformation permettant d'alimenter MS-Project à partir du modeleur. Nous finissons par présenter le mode opératoire, la manière dont ces différents composants peuvent être utilisés.

8.2. MS-Project

MS-Project est un outil de planification de projet distribué par Microsoft. Cet outil est très utilisé par les chefs de projet dans le cadre de projets industriels. Il fournit un ensemble de fonctionnalités pour planifier les projets et en effectuer le suivi. Il permet de définir la hiérarchie des tâches du projet et leur ordonnancement via des liens de dépendance. Il permet également de spécifier un certain nombre d'informations (coûts, disponibilités) sur les ressources humaines et matérielles pouvant être utilisées lors d'un projet. Chaque tâche peut être planifiée. On peut ainsi estimer sa durée, fixer une date d'échéance, etc. En recoupant la durée des tâches avec le coût des ressources qui leur sont affectées, on peut

prévoir le coût du projet et contrôler son évolution. Enfin, un certain nombre de facilités sont fournies pour obtenir différents rapports sur les projets et les ressources.

Le méta-modèle présenté Figure 117 est une partie du méta-modèle complet de MS-Project (pour des raisons de place et de lisibilité nous ne montrons pas l'ensemble des propriétés de chacune des classes). Il a été mis au point d'une part en utilisant le progiciel, et d'autre part en étudiant le format de stockage des informations dans une base de données. Le projet (*MSPProject*) en est l'entité centrale. Un projet peut être composé de ressources (*MSPResource*), de tâches (*MSPTask*), de liens (*MSPLink*) ainsi que de calendriers (*MSPCalendar*). Il contient un certain nombre d'information relatives à sa définition (le nom de l'auteur, de l'entreprise), ainsi que des données calculées (date de fin).

La tâche est l'unité de travail. Elle peut être composée par des sous-tâches. Elle contient des données prévisionnelles (*startDate*, *finishDate*, *cost*), des données que l'on fixe (*constraintDate*), et des données effectives (*actStart*, *actFinish*, *actCost*). Un lien (*MSPLink*) indique une dépendance entre deux tâches. Cette dépendance peut être de plusieurs types (fin à début, début à début, fin à fin, début à fin).

Les ressources peuvent être humaines (*MSPHumanResource*) ou matérielles (*MSPMachine*). Une ressource humaine peut être disponible (*MSPAvailability*) un certain nombre d'heure entre deux dates. Toutes les ressources ont des coûts (*MSPResourceRate*). On peut distinguer le coût par utilisation (*costPerUse*), le coût standard (*stdRate*) et le coût en heure supplémentaire (*ovtRate*). Ces coûts peuvent varier d'une période à l'autre. Enfin, les disponibilités des ressources obéissent à un calendrier (*MSPCalendar*). Un calendrier indique les plages horaires (*MSPWorkPeriod*) des jours ouvrés. Le calendrier d'une ressource particulière peut être basé sur un calendrier de base (équipe de nuit, standard, 24 heures, etc.).

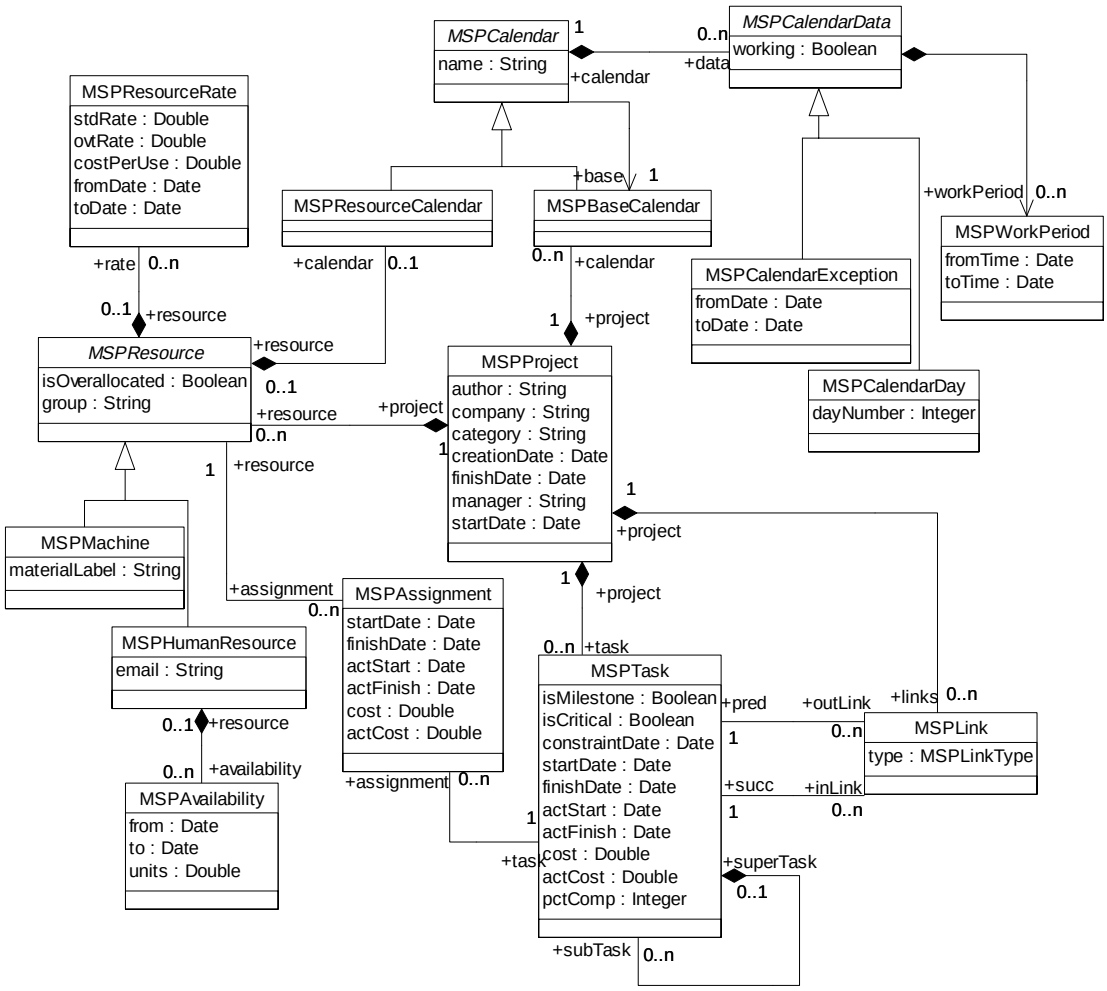


Figure 117. Le méta-modèle de MS-Project

8.3. Réalisations

Une fois que nous avons défini le méta-modèle de MS-Project, il a été facile de l'intégrer dans Scriptor-T. Nous avons alors pu définir les règles de transformation d'un modèle de processus décrit dans le modeleur Sodifrance en un projet MS-Project.

```
PMM -> MSP (
        ONE_PASS_ONLY = true;
):

// Transformation des processus en projets
PMMProcessPackage(p1) name(p1,name) ->

        MSPProject(p2) name(p2,name) title(p2,name) project(w2,p2) _coref(p1,p2)
        author(p2,"Soft-Maint") calcActCosts(p2,true) company(p2,"Soft-Maint")
        slackLimit(p2,0) currencyDigit(p2,2) currencyPosition(p2,"3") currencySymbol(p2,"F")
        newAreEffortDriven(p2,true) defFixCostAccrual(p2,"3") minutesPerDay(p2,480)
        minutesPerWeek(p2,2400) defOvtRate(p2,0.0) defStdRate(p2,0.0) defTaskType(p2,"0")
        durEntryFmt(p2,"7") honorConstraints(p2,true) multCriticalPaths(p2,false) calName(p2,"Standard")
        isResPool(p2,false) schedFrom(p2,"1") splitInProgress(p2,true) spreadActCosts(p2,false)
        spreadPctComp(p2,false) taskUpdatesRes(p2,true) workEntryFmt(p2,"5") calcSubAsSummary(p2,true)
        weekStartDay(p2,"1") fyStartMonth(p2,"1") fyUseStartYr(p2,true) daysPerMonth(p2,20)
        newTaskEst(p2,true) showEstDur(p2,true) expandTimephased(p2,false) editedDate(p2,false)
        editedDur(p2,false) editedNum(p2,false) editedFlag(p2,false) editedCode(p2,false) editedText(p2,false)
        edited(p2,false) readOnly(p2,"0") readWrite(p2,"0") readCount(p2,"0") locked(p2,"0") ;

// Transformation des étapes actives en tâches
PMMActionStep(as) name(as,name) linktoContainerPackage(as,pec) package(pec,p1) _coref(p1,p2) ->

        MSPTask(t) name(t,name) project(t,p2) _coref(as,t)
        acwp(t,0.0) bcwp(t,0.0) bcws(t,0.0) durVar(t,4800) finishVar(t,0) startVar(t,0) isOverallocated(t,false)
        ovtWork(t,0.0) vac(t,0.0) regWork(t,0.0) numObjects(t,0) totalSlack(t,0) hasLinkedFields(t,false)
        isMilestone(t,false) isCritical(t,true) isSummary(t,false) isSubProj(t,false) isMarked(t,false)
        ignoreResCal(t,false) isRolledUp(t,false)isFromFinishSubProj(t,false) barIsHidden(t,false)
        isRecurring(t,false) isRecurringSummary(t,false) isExternal(t,false) isEffortDriven(t,true)
        isCollapsed(t,false) hasNotes(t,false) isReadonlySubproj(t,false) levelingCanSplit(t,true)
        levelingAdjustsAssn(t,true) durIsEst(t,false) freeSlack(t,0) dur(t,4800) durFmt(t,53) actDur(t,0)
        remDur(t,4800) baseDur(t,0) baseDurFmt(t,39) constraintType(t,"0") levelingDelay(t,0)
        levelingDelayFmt(t,8) priority(t,500) pctComp(t,0) pctWorkComp(t,0) type(t,"0") fixedCostAccrual(t,3) ;

// Pour les activités: is Summary:=true
PMMActivity(a) _coref(a,t) ->

        isSummary(t,true) isEffortDriven(t,false) type(t,"1") ;

// Création du lien subTask entre une tâche et son parent
PMMActivityGraph(g) linktoSuperStep(g,asg) activity(asg,a) linktoContainedSteps(g,gsc)
        step(gsc,s) _coref(a,t1) _coref(s,t2) ->

        subTask(t1,t2) ;


// Transformations des transitions en liens
PMM_TransitionStepFollowing(tsf) step(tsf,s) !PMMEndStep(s) _coref(s,t) transition(tsf,tr)
        linktoFormerSteps(tr,tsp) step(tsp,sp) _coref(sp,st) linktoContainerPackage(tsf,pec)
        package(pec,p1) _coref(p1,p2) ->

        MSPLink(l) type(l,"1") _coref(tsf,l) succ(l,t) project(l,p2) pred(l,st) ;
```

Figure 118. Règles de transformation de PMM vers MSP

Un processus PMM est transformé en un projet MS-Project. Toutes les tâches et activités (toutes les instances des concepts héritant de *PMMActionStep*) sont transformées en tâches MS-Project (*MSPTask*). La hiérarchie entre les tâches est conservée. Il en est de même pour l'ordonnancement des tâches. Une transition (*PMMTransition*) va donner lieu à la création d'un à plusieurs liens (*MSPLink*). En effet, un lien MS-Project indique une

dépendance entre deux tâches, alors qu'une transition dans le méta-modèle PMM indique un ordonnancement temporel entre N étapes sources et M étapes cibles. A partir d'une seule transition, on va donc créer N*M liens.

Ces règles de transformation permettent le transfert d'information du modeleur de processus vers l'outil de planification. La production du workflow à partir du modèle de processus se fait grâce à Scriptor-G, de la façon qui a été évoquée dans la section précédente. Enfin, l'outil de planification et le workflow interagissent via un certain nombre de composants exécutables, des Flowbeans (voir la partie consacrée à FlowMind dans la section précédente), développés de façon ad hoc (voir Figure 119). Les invocations de ces Flowbeans sont intégrées dans le workflow à la génération. Avant chaque étape, un Flowbean se charge de mettre à jour la planification, afin d'indiquer la date réelle de début de l'étape. De même, à la fin de chaque étape, un autre Flowbean signale à l'outil de planification que l'étape s'est terminée à telle date. Enfin, avant chaque tâche manuelle, le workfow interroge la planification pour savoir à qui adresser la tâche (potentiellement plusieurs personnes) et quelles sont les contraintes de temps à respecter. Dans le cas où aucune personne n'a été désignée pour réaliser la tâche, le responsable de la planification est averti de ce manque. Chaque étape du processus produit donc des étapes relativement complexes au niveau du workflow.

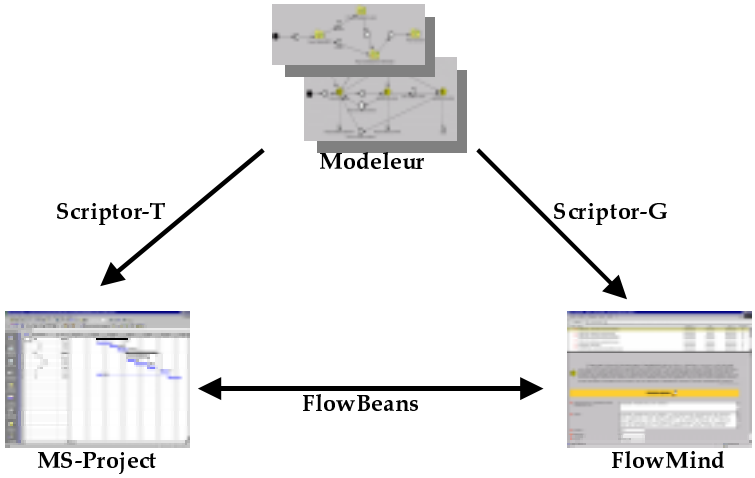


Figure 119. Les différents composants de la solution

Ce prototype a été développé pour démontrer la faisabilité et l'intérêt de la solution. Aussi, il demeure quelques problèmes non traités. Il s'agit principalement des restrictions dues à MS-Project qui ne peut prendre en compte les boucles et les branches conditionnelles d'un processus. Une solution consisterait à extraire du modèle de processus le scénario d'exécution le plus courant, puis à l'adapter dynamiquement lors de l'exécution en fonction du contexte.

8.4. Mise en œuvre

Nous allons maintenant illustrer les principes définis dans les paquetages précédents en les mettant en œuvre sur un exemple concret. Nous avons choisi d'appliquer l'intégration de la définition, de la planification et de l'exécution sur un processus de développement logiciel fortement inspiré du RUP (Rational Unified Process). La première tâche à réaliser consiste à définir le processus à l'aide du modeleur. Nous avons donc spécifié la séquence de tâches relatives à une itération (Figure 120 et Figure 121).

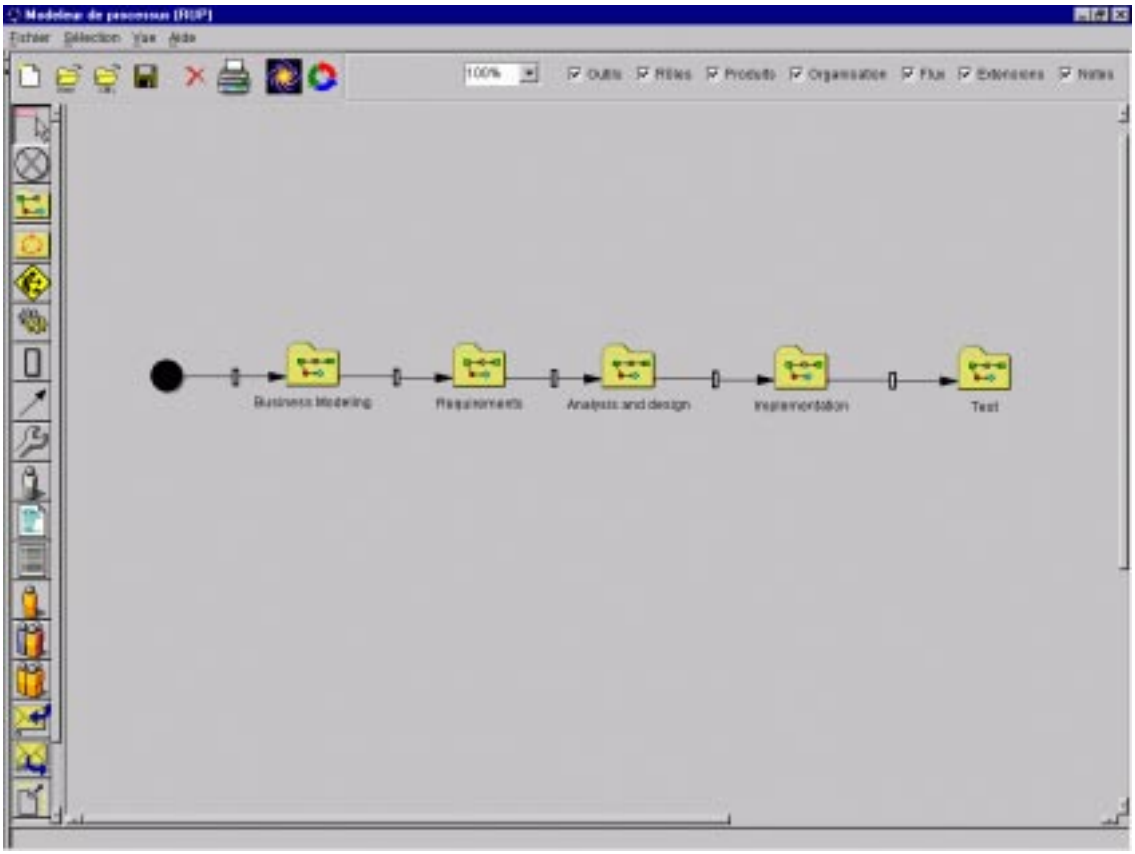


Figure 120. Les principales étapes d'une itération

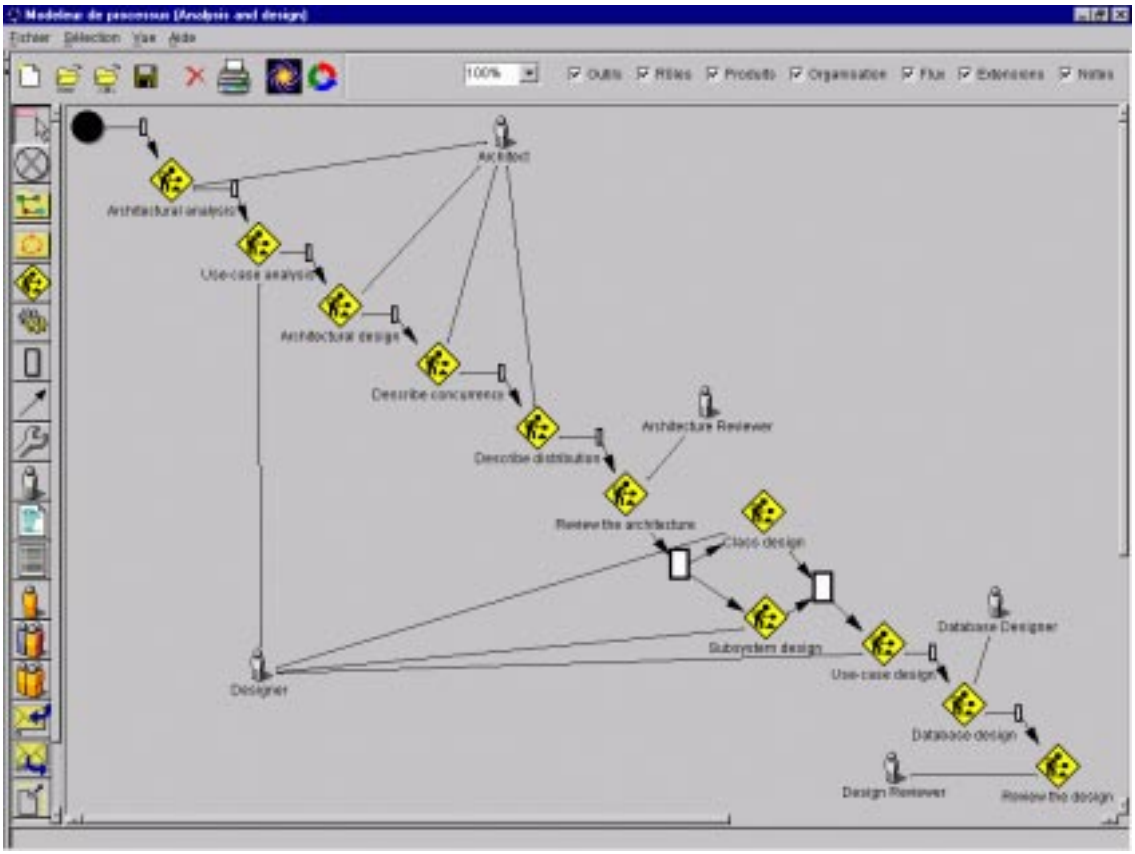


Figure 121. Décomposition en tâches de l'étape analyse et conception

Une fois que le processus est complètement défini, il peut être mis en œuvre dans le cadre d'un projet. Sur un projet, on distingue donc deux vues particulières, celle du chef de projet et celle des réalisateurs. Au démarrage du projet, le chef de projet va réaliser une planification initiale. Pour cela, il va alimenter MS-Project avec le processus défini par le responsable méthode. Il disposera ainsi du cadre de base pour spécifier les affectations de ressources, les charges et les délais (Figure 122).

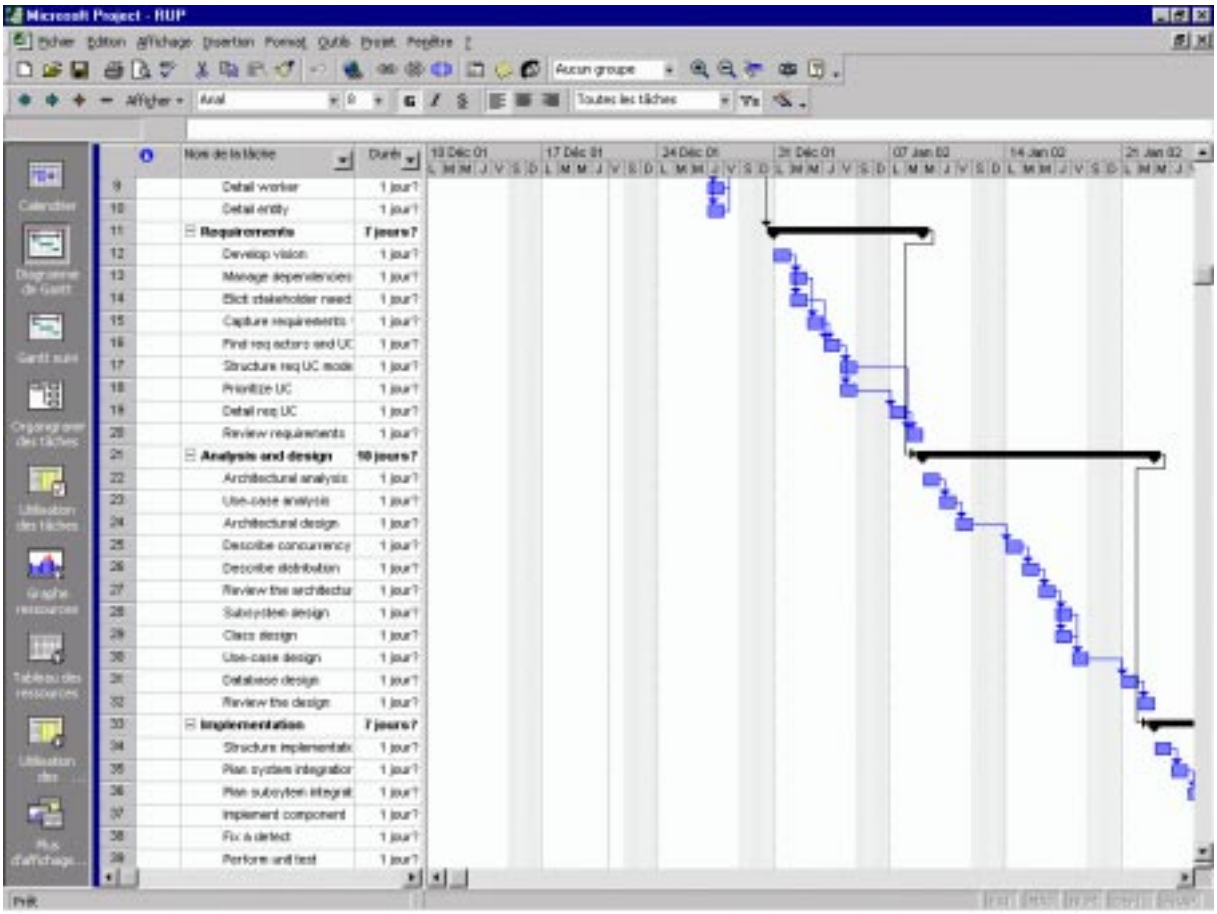


Figure 122. Résultat de la transformation

La transformation permet donc de récupérer dans l'outil de planification l'ensemble des tâches, leur hiérarchie et leurs dépendances. L'intervention du chef de projet se limite alors à estimer les durées de chacune de ces tâches et à spécifier les affectations. Ainsi, on définit la planification initiale du projet (Figure 123).

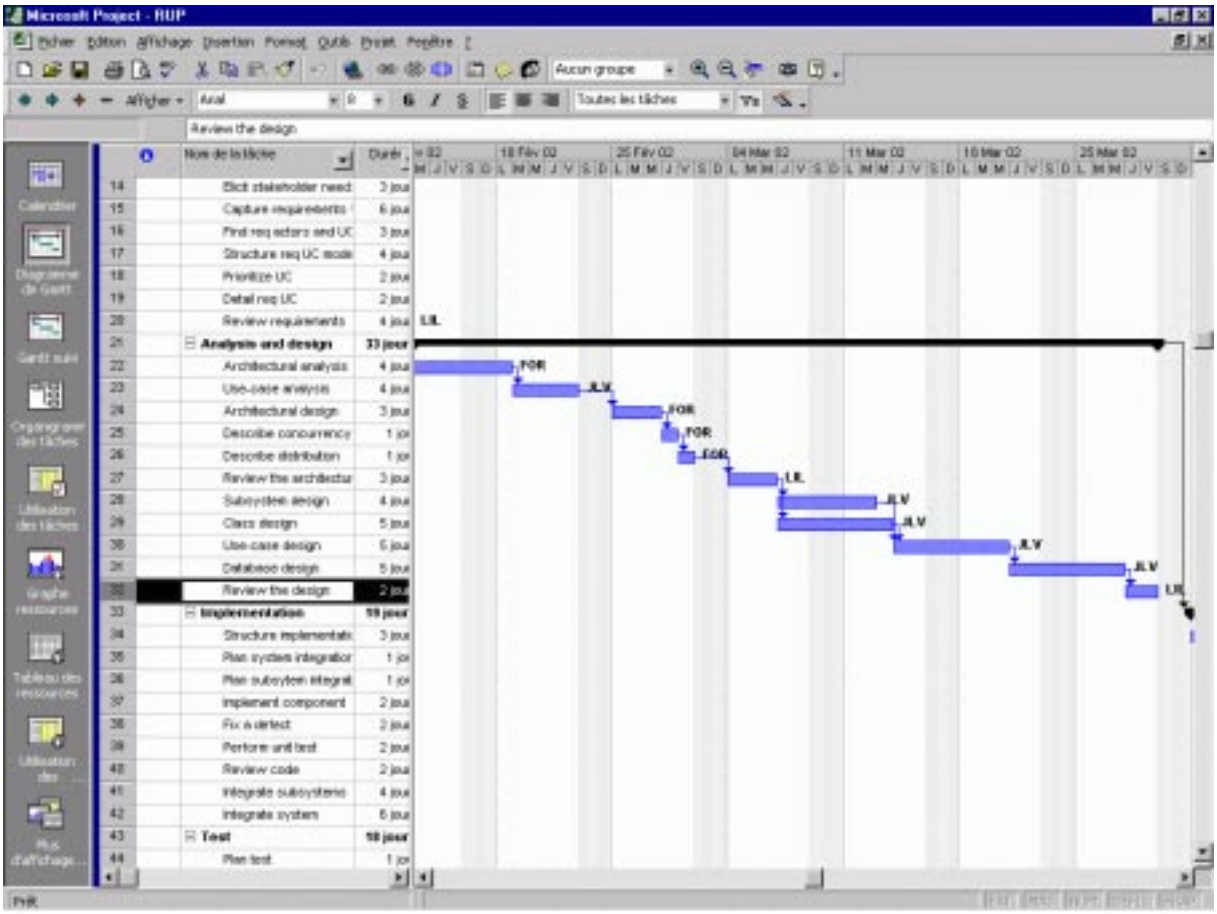


Figure 123. Planification initiale

Une fois que la planification initiale a été spécifiée, on peut démarrer l'exécution du projet, c'est-à-dire que l'on peut activer le workflow correspondant. Celui-ci prend en compte les affectations de ressources définies dans la planification, et il met à jour un suivi qui permet de constater les écarts de délais entre la planification du processus et son exécution (Figure 124). Cette mise à jour est réalisée de façon totalement transparente pour les utilisateurs.

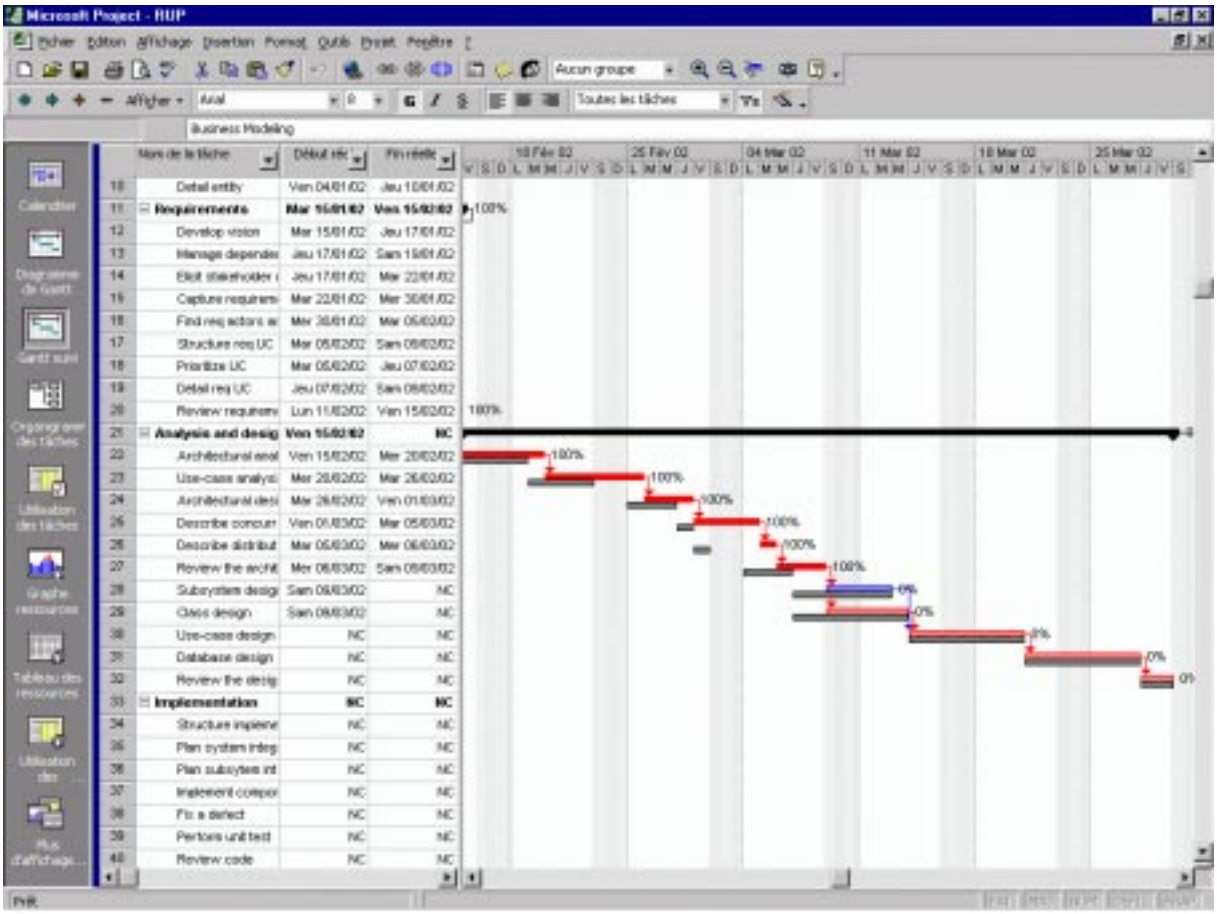


Figure 124. Suivi du processus par rapport à la planification initiale

8.5. Conclusion

Dans ce prototype, on a identifié trois outils intervenant à des moments différents dans le cycle de vie du processus. Le modeleur de processus permet de décrire le modèle. MS-Project permet de planifier une exécution du processus. Enfin, le moteur de workflow permet de contrôler l'exécution proprement dite. Chacun de ces outils répond à des besoins différents. Ils proposent donc chacun un formalisme différent. Toutefois, nombre de parties communes existent entre ces différents formalismes. L'intérêt de la méta-modélisation est donc tout d'abord de faciliter leur identification. En représentant chacun des formalismes de façon uniforme, il est relativement aisé d'établir des correspondances. Des techniques de transformation vont ensuite pouvoir être utilisées pour acheminer l'information entre ces différents outils, et donc entre les différents acteurs ayant partie prenante dans le cycle de vie du processus.

Conclusion

Dans cette partie nous avons présenté un certain nombre de réalisations industrielles. Elles nous ont permis d'appliquer les techniques de méta-modélisation dans le cadre de la gestion des processus. Nous avons ainsi pu valider leur pertinence dans ce domaine. La plupart des travaux que nous présentés dans cette partie peuvent se situer dans le cadre du MDA. En effet, notre démarche consiste à tenter de séparer au maximum les aspects purement métier des préoccupations relatives au déploiement du processus sur un quelconque moteur d'exécution. Notre ambition première était de préserver les processus standards de Sodifrance des spécificités rencontrées sur chacun des sites. L'adaptation du processus se fait au travers d'extensions du méta-modèle originel dédiées à paramétrer son intégration dans l'environnement logiciel cible ou à prendre en compte des besoins fonctionnels supplémentaires. Cette stratégie est donc relativement similaire à l'utilisation de profils pour UML. La production d'un processus exécutable est ensuite automatisée grâce aux techniques de transformation de modèles et de génération de code. Une fois que les extensions ont été définies et que les services de génération ont été développés, la production des processus exécutables peut alors être industrialisée. Le nombre d'interventions humaines se trouve ainsi grandement minimisé, ce qui permet une plus grande efficacité et une qualité constante du résultat. Enfin, des modèles de processus même incomplets peuvent être très rapidement testés en situation réelle. Les services de génération se chargent alors de combler les lacunes du modèle par un certain nombre de choix arbitraires.

Du point de vue purement industriel (cette thèse faisant l'objet d'une bourse CIFRE, il nous semble important de souligner cette partie), ces travaux ont débouché sur un certain nombre de résultats opérationnels. Tout d'abord, Sodifrance commercialise aujourd'hui une offre autour de l'ingénierie de processus basée sur les différents composants que nous avons présentés précédemment : le modeleur de processus, Scriptor-T et Scriptor-G. Un certain nombre de prototypes ont ainsi pu être réalisés pour des grands groupes (Alcatel, Thalès), principalement dans le domaine du processus de développement logiciel. Les processus métier de Sodifrance, qui constituaient la cible initiale, ont également été traités. On peut principalement citer à ce sujet le processus de tierce maintenance applicative que nous avons déployé. Celui-ci est aujourd'hui utilisé quotidiennement par une cinquantaine de personnes, et ce depuis plus d'un an. Enfin, ces travaux ont permis à Sodifrance de contractualiser un partenariat avec un éditeur de workflow, Akazi. Nos travaux de méta-modélisation ont ainsi été réutilisés pour la spécification de leur propre outil de modélisation de processus.

Partie III
GENERALISATIONS ET PROPOSITIONS

Introduction

Dans la partie précédente, nous avons présenté un certain nombre de réalisations. Nous avons ainsi pu mettre en pratique une partie de nos idées sur l'ingénierie de processus sur des cas concrets. Cela nous a également permis d'identifier quelques problématiques. Dans cette dernière partie, nous nous attachons à les expliciter, ainsi qu'à leur apporter des réponses conceptuelles, toujours basées sur les techniques de méta-modélisation. Nous nous intéressons donc aux quatre points suivants.

Le premier point que nous abordons est l'existence d'un noyau de concepts caractéristiques des méta-modèles de processus. Dans la première partie, nous avons étudié différents contextes d'utilisation du concept de processus, ainsi que différents formalismes de description. Nous avons pu établir, de façon informelle, l'existence d'un nombre important de similarités. A partir de la représentation homogène de ces formalismes, sous la forme de méta-modèles, nous tentons d'expliciter de façon plus précise ces différents points de convergence.

De nombreux auteurs ont souligné l'existence d'un couplage fort entre la description du processus et celle des produits qu'il utilise. Dans la plupart des cas, ces remarques n'ont pas donné lieu à la définition de relations explicites entre les univers produit et processus, principalement à cause du manque de formalismes sur lesquels baser ces spécifications. Nous étudions la manière dont de telles relations peuvent être explicitées à l'aide des techniques de méta-modélisation, en basant notre analyse sur un exemple concret, le couplage de PMM et d'UML.

La plupart des formalismes de description de processus que nous avons étudiés se limitent à la seule définition du processus. Ils ne fournissent pas de mécanismes permettant de représenter l'exécution d'un processus. Nous nous intéressons à la façon dont ces deux aspects peuvent être définis et intégrés au sein d'un même formalisme.

Enfin, nous terminons cette dernière partie en traitant de l'expression des règles spécifiant l'exécution de processus. La prise en compte d'aspects dynamiques dans un méta-modèle de processus permet de représenter l'exécution d'un processus. L'ajout des règles spécifiant cette exécution permettrait de le simuler, voire de l'exécuter.

9. Caractérisation des méta-modèles de processus

9.1. Introduction

Beaucoup de formalismes de définition de processus sont apparus récemment (SPEM, EDOC, BPML, ebXML, WSFL, XLANG). Des travaux supplémentaires, comme la spécification de l'OMG pour la définition de workflow, sont actuellement en cours. Nous sommes donc confrontés à une multiplication des formalismes. Chacun adopte une terminologie particulière et définit des concepts spécifiques en fonction des objectifs du méta-modèle, du domaine d'intérêt ou de choix de représentation. Toutefois, dans l'ensemble des méta-modèles de processus que nous avons étudié, ainsi que dans PMM, nous pouvons mettre en évidence un certain nombre de structures similaires. Pour faciliter le développement de nouveaux méta-modèles de processus et le transfert d'informations d'un formalisme à l'autre, il nous paraît pertinent de tenter d'expliciter l'existence d'un ensemble de concepts communs.

Nos différentes études et nos travaux nous ont amené à dégager un certain nombre d'aspects qui nous semblent génériques. Nous considérons qu'un processus est un ensemble organisé d'activités dont la réalisation entraîne l'utilisation et la production d'objets. Notre vision est donc très proche de celle que l'ISO expose à travers sa définition (« *Set of interrelated or interacting activities which transforms inputs into outputs* » [37]). Un méta-modèle de processus peut être caractérisé par les notions suivantes : le processus, les activités, les structures de contrôle (qui définissent l'organisation des activités) et les objets (qui sont utilisés et produits par les activités).

Dans ce chapitre, nous allons donc tenter de justifier et d'expliciter cette vision. Nous commençons par aligner les méta-modèles étudiés dans la première partie ainsi que PMM au travers des aspects énoncés ci-dessus. Ce travail est grandement facilité par l'utilisation d'un seul et unique formalisme pour les représenter. Pour chacun des aspects, nous comparons les différents mécanismes de représentation. Nous terminons en proposant un méta-modèle générique de processus en tenant compte des diverses observations que nous avons pu faire.

9.2. Le concept de processus

Le concept de processus n'est pas défini de façon explicite dans tous les formalismes. Ainsi, ARIS, TOVE, PIF et PSL ne l'intègrent pas. Il peut alors être identifié avec une activité macroscopique ou avec le modèle. Quand il apparaît, il peut être désigné par différents termes : processus, scénario, plan, modèle de flux, graphe d'activité ou comportement (voir Table 1).

Table 1. Alignement des concepts de processus

Formalisme	Concepts
ARIS	
BPML	Process Abstract
CPR	Plan
ebXML	MultipartyCollaboration BinaryCollaboration
EDOC	BusinessProcess CompoundTask
IDEF3	Scenario
PIF	
PMM	ProcessPackage
PSL	
SPEM	Process
TOVE	
UML	Activity Graph
WPDL	Workflow Process
WSFL	FlowModel
XLANG	Behavior

Une des caractéristiques partagée par la plupart des définitions est le rôle de conteneur du processus. Ainsi, dans SPEM, le processus spécialise la notion de paquetage. Sans avoir connaissance de ces travaux, nous avions préalablement proposé une organisation similaire dans PMM. C'est également le cas dans BPML où le processus contient les autres éléments. CPR définit même la décomposition possible d'un plan en sous-plans. Dans la plupart des cas, la décomposition d'un processus se fait par l'intermédiaire d'une activité. Ainsi, EDOC, IDEF3, WPDL ou encore BPML définissent un lien permettant à une activité de réutiliser un processus.

Dans de nombreux formalismes, on retrouve une relation particulière entre le processus et l'activité. Cette relation définit l'activité de plus haut niveau du processus. C'est le cas d'UML, BPML ou de XLANG. Le même principe se retrouve dans SPEM où le processus est gouverné par un cycle de vie.

La définition d'un processus peut être contextuelle à un élément externe. C'est le cas d'UML où le graphe d'activité est rattaché à un classifieur. C'est également le cas de WSDL et XLANG car, même si cela n'apparaît pas explicitement dans le méta-modèle, le processus est destiné à être intégré dans un service Web. Les autres formalismes définissent les processus comme des entités indépendantes.

9.3. Les activités

Les activités constituent le cœur du processus. Elles définissent le travail à réaliser. Comme le montre le tableau suivant, elles sont présentes dans l'ensemble des formalismes étudiés. A nouveau, les termes utilisés varient d'un formalisme à l'autre (voir Table 2).

Table 2. Alignement des concepts d'activité

Formalisme	Concepts
ARIS	Function
BPML	Activity et toutes ses spécialisations
CPR	Action
ebXML	BusinessActivity CollaborationActivity BusinessTransactionActivity
EDOC	Activity

IDEF3	UOB
PIF	Activity
PMM	Step Activity Task
PSL	Activity
SPEM	WorkDefinition Activity Phase Iteration Lifecycle
TOVE	Activity
UML	ActionState SubactivityState FinalState PseudoState
WPDL	Workflow Activity
WSFL	Activity
XLANG	Process et toutes ses spécialisations ActionBase

Comme on peut le voir, le concept d'activité a souvent donné lieu à la définition de plusieurs autres concepts. Dans la plupart des cas, ceux-ci ont pu être organisés dans un graphe d'héritage. Il existe fréquemment une distinction entre activité composée et activité atomique. On la retrouve dans BPML, UML, EDOC, XLANG, ebXML et nous l'avons également introduite dans PMM. Dans SPEM, plusieurs niveaux d'activités composées ont été identifiés : *Iteration*, *Lifecycle*, *Phase* et *WorkDefinition*. Dans certains cas, l'activité composée induit un ordonnancement spécifique de ses sous-activités. On peut trouver de tels exemples dans BPML (*All*, *Sequence*, *Foreach*) et dans XLANG (*All*, *Switch*, *While*).

Des mécanismes différents sont utilisés pour définir des activités composées. Il peut s'agir d'une relation de composition, les sous-activités font alors partie intégrante de l'activité composée. Il peut également s'agir d'une simple référence, les sous-activités pouvant alors être intégrées dans plusieurs activités composées différentes. Enfin, la décomposition de l'activité peut être définie par un processus, l'activité composée définissant alors un lien vers un processus.

Les activités atomiques définissent les opérations de transformation de base du système d'information. Elles peuvent également faire l'objet de spécialisations. Ainsi, dans PMM nous avons pu différencier l'activité automatique de l'activité manuelle. BPML identifie, entre autres, la production et la consommation d'un message, l'invocation et l'implémentation d'une opération.

Une des différences qu'on peut parfois trouver entre activité complexe et activité atomique réside dans le fait que les relations vers les objets ne sont définies que sur les activités atomiques. Ce sont les seules activités atomiques qui utilisent et produisent des objets. C'est le cas par exemple de EDOC, de BPML, de XLANG ou encore de ebXML. Nous avons également retenu cette solution dans PMM. Ce n'est toutefois pas le cas de PIF, de PSL ou de SPEM.

Enfin, on peut trouver certaines activités définissant des mécanismes particuliers tels que le point de départ, le point de fin, la synchronisation ou la création de branches parallèles. Ces activités ne définissent pas à proprement parler un travail à réaliser. Elles spécifient des mécanismes de contrôle (voir le paragraphe suivant).

9.4. Les structures de contrôle

Les structures de contrôles décrivent les dépendances entre activités. Elles déterminent l'exécution du processus. Elles permettent de définir (alliées aux préconditions) le moment où une activité doit être réalisée. Elles sont donc particulièrement importantes dans sa définition. Dans la théorie de la coordination ([17], [55]), les dépendances suivantes ont été identifiées :

- La décomposition activités/sous-activités
- Les contraintes de simultanéité : séquencement, synchronisation
- Les relations producteur/consommateur
- Le partage d'une même ressource par deux activités

Comme nous l'avons vu précédemment, la décomposition d'une activité en sous-activités se fait par l'intermédiaire de relations de composition ou de référence.

Les contraintes de simultanéité peuvent être représentées de différentes façons. D'une part, il y a fréquemment la définition du concept de transition, indiquant une dépendance entre deux activités. On la retrouve dans ebXML, dans UML, dans WPDL, dans WSFL

(*ControlLink*), dans SPEM (*precedes*), dans PIF (*Successor*) et dans IDEF3 (*PrecedenceLink*). Nous avons également utilisé ce mécanisme dans PMM. De plus, comme on l'a vu certaines activités atomiques pouvaient introduire des comportements spécifiques au sein du graphe d'activité. Enfin, comme nous l'avons déjà évoqué, certaines activités composées définissent un flux de contrôle entre leurs sous-activités de manière intrinsèque.

Les dépendances de type producteur/consommateur sont le plus souvent implicites. On peut déterminer quelles activités utilisent un objet, et lesquelles le produisent ou le modifient. Ce n'est toutefois pas le cas dans EDOC ou toutes les dépendances entre activités sont des connections entre ports, et donc des relations de producteur à consommateur. UML permet également de définir les flux d'objets entre activités (*ObjectFlowState*), ce flux pouvant donner lieu à synchronisation. WSFL fournit aussi un mécanisme pour préciser explicitement le flux des données entre activités par l'intermédiaire des *DataLinks*. Ceux-ci ne définissent pas de synchronisation des tâches. Ils viennent se surexposer à la structure de contrôle.

Cette explicitation du flux de données nous semble importante. En effet, dans bien des cas, une relation de précédence entre deux activités masque une dépendance de type producteur/consommateur. Ne pas l'expliciter peut rendre plus complexe la réingénierie du processus, car on manque d'information pour décider de façon argumentée du bien-fondé de la relation de précédence.

Les dépendances dues au partage d'une même ressource sont rarement définies. Une des extensions de PSL introduit le concept d'interférence. Une interférence entre deux occurrences d'activités indique qu'elles partagent une ressource. De même l'ontologie de ressources définies dans TOVE définit si deux activités peuvent utiliser simultanément une ressource donnée.

Enfin, TOVE ne définit pas explicitement de dépendance entre les activités. Chaque activité a un état d'activation et un état résultat. Une activité A précède une activité B si l'état résultat de A correspond à l'état d'activation de B.

Le tableau suivant décrit les principaux mécanismes de définition du flux de contrôle. Nous n'y avons pas intégré les relations de composition des activités.

Table 3. Alignement des concepts spécifiant la structure de contrôle

Formalisme	Concepts
ARIS	
BPML	All Sequence Foreach

CPR	
ebXML	Transition Join Fork
EDOC	DataFlow
IDEF3	PrecedenceLink Junction
PIF	Successor
PMM	Transition
PSL	
SPEM	Precedes
TOVE	
UML	Transition ObjectFlowState PseudoState SynsState
WPDL	TransitionInformation
WSFL	ControlLink Join
XLANG	Sequence While All

9.5. Les objets

Dans PMM, nous avons distingué trois ensembles distincts d'objet : les documents, les outils et la structure organisationnelle. Cela nous semble pertinent dans le contexte d'utilisation de PMM, mais ce n'est pas le cas pour tous les domaines d'application. Par exemple, BPML, qui est dédié à la définition de processus inter-entreprises, ne définit que les notions de messages et de participants. Dans le domaine des processus à base de composition de services Web, WSFL et XLANG manipulent principalement des opérations WSDL et des données. Si on fait le parallèle avec l'ingénierie des connaissances, on peut comparer processus, activités et structures de contrôle au modèle de raisonnement, alors que les objets correspondent au modèle du domaine. Alors que les différents modèles de raisonnement sont relativement proches, les modèles de domaine sont extrêmement variés.

Table 4. Alignement des concepts d'objet

Formalisme	Concepts
ARIS	Output OrganizationalUnit InformationObject Application Software
BPML	Message Participant
CPR	Resource Actor
ebXML	BusinessDocument BusinessPartnerRole AutorizedRole
EDOC	Artifact Performer
IDEF3	Object
PIF	Object Agent
PMM	Product

	ProcessData Role Tool
PSL	Object
SPEM	WorkProduct InformationElement ProcessRole Guidance
TOVE	Resource
UML	Classifier ClassifierInState
WPDL	Relevant Data Workflow Participant Workflow Application
WSFL	Message ServiceProvider Operation
XLANG	CorrelationSetDecl Service Operation

Les deux relations principales entre activités et objets sont les relations d'utilisation et de production. On les retrouve sous différentes formes dans tous les formalismes. A ces deux relations s'ajoute souvent la relation entre l'activité et ses réalisateurs. Le plus souvent, le réalisateur est unique. Cette relation peut être ramenée à la relation d'utilisation si l'on ne s'intéresse qu'au coût d'une activité comme c'est le cas en gestion. Le réalisateur n'est qu'une des ressources nécessaires à l'activité, au même titre que les matières premières ou l'énergie. Elle définit toutefois un rôle spécifique par rapport à l'activité. C'est l'objet réalisateur qui effectue les opérations de transformations élémentaires (même s'il les

délègue ensuite à d'autres objets). Il faut donc lui signaler que l'activité a démarré (invocation du composant logiciel, envoi d'un signal, etc.). C'est également souvent cet objet qui signale la fin de la réalisation.

Dans de nombreux cas, une définition précise des objets demande l'intégration d'un méta-modèle dédié. Ainsi, WPDL prévoit la possibilité de relations entre modèles de processus et modèles organisationnels. En effet, un méta-modèle tel que celui du WfMC propose un mécanisme de déclaration des participants (les entités organisationnelles auxquelles pourront être affectées des tâches) extrêmement basique. Dans le cas de nombreuses implémentations, ce mécanisme se limite à la désignation de groupes ou de personnes physiques. Un tel modèle peut donc se révéler insuffisant dans le cadre de requêtes plus complexes. Par exemple, on peut vouloir affecter une tâche au responsable hiérarchique de la personne qui a réalisé la tâche précédente. Dans de tels cas, il faut pouvoir définir précisément les statuts hiérarchiques de toutes les personnes physiques susceptibles de prendre part au processus, leurs appartenances ainsi que les relations qui les lient. On peut même souhaiter spécifier leurs droits ou leurs compétences. Comme l'ajout de tels concepts dans le méta-modèle de processus l'alourdirait, la solution préconisée est l'intégration d'un modèle d'organisation externe. Les concepts d'objets introduits dans les méta-modèles de processus indépendants restent donc le plus souvent primaires.

9.6. Quelques mécanismes supplémentaires

En plus de ces différents concepts, les méta-modèles de processus peuvent intégrer un certain nombre de mécanismes. Nous allons nous pencher plus particulièrement sur deux d'entre eux :

- La réutilisation
- La spécification des contraintes

La réutilisation consiste à pouvoir intégrer dans un processus des éléments définis dans d'autres contextes. On peut distinguer la réutilisation d'éléments statiques (rôles, produits) de la réutilisation d'éléments dynamiques comme les activités ou les processus. Dans le premier cas, l'extension de paquetage est une solution suffisante. C'est par exemple celle qui est mise en œuvre dans SPEM et PMM. Par contre, la réutilisation d'éléments dynamiques est bien plus complexe. En effet, la définition d'un processus ou d'une activité peut intégrer la définition d'artefacts locaux. Intégrer cette définition nécessite donc d'unifier ces concepts locaux avec ceux du contexte d'utilisation. C'est le problème de l'unification, pour lesquels SPEM propose actuellement un renommage manuel des artefacts de la portion de processus à intégrer. Ce mécanisme de renommage a par exemple été utilisé par Robin Milner dans CCS, tout comme par Tony Hoare dans CSP (voir la première partie). Ce mécanisme est non seulement nécessaire pour la réutilisation d'activités ou de processus, mais également pour l'intégration de composants externes. Ainsi WSFL inclut le concept de *Map* qui établit des correspondances entre les messages acheminés vers une activité, et la structure de données (qui est également un message) interne à l'activité. Dans EDOC, cette

correspondance est établie entre les ports des composants de processus qui cherchent à communiquer.

Un autre mécanisme que nous jugeons important, voire même indispensable, dans un méta-modèle de processus est le langage de spécification de contraintes. Ces contraintes peuvent apparaître à différents endroits : dans les pré- et post-conditions des activités ou dans les structures de contrôle conditionnées (boucles, alternatives). Ainsi UML utilise un langage pour la définition des gardes qui est réutilisé par SPEM pour les pré-conditions et les objectifs. Ce langage permet de spécifier des contraintes sur les états des produits de la façon suivante « myProduct **in** myState ». Nous avons également développé notre propre langage pour PMM. Enfin, des formalismes basés sur XML (BPML et WSDL) proposent d'utiliser XPath. XPath n'est pas à proprement parler un langage de contraintes, mais il permet de désigner un élément dans un document XML.

9.7. Définition d'un noyau de méta-modèle de processus

A partir des différentes observations qui ont été faites dans les chapitres précédents, nous sommes maintenant en mesure d'expliciter notre vision d'un noyau de méta-modèle de processus (Figure 125). Nous n'avons pas souhaité le définir comme le plus petit dénominateur commun des différents formalismes étudiés. Un tel méta-modèle aurait été des plus sommaire. Nous avons plutôt cherché à établir un compromis en définissant un méta-modèle utilisable tout en ne retenant que les structures les plus courantes. Il est d'ailleurs assez significatif que ce méta-modèle soit très proche de PIF, qui définit également un formalisme de représentation de processus générique mais non minimal.

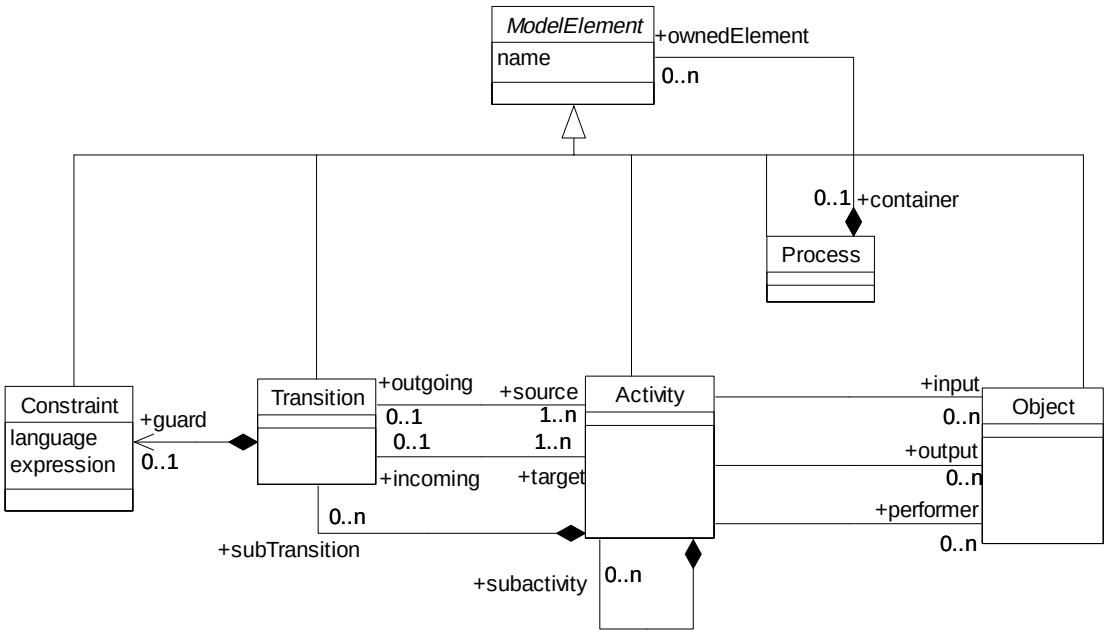


Figure 125. Un noyau de méta-modèle de processus

A l'intérieur de ce méta-modèle, on retrouve les notions de processus (*Process*), d'activité (*Activity*), de structures de contrôle (*Transition*) et d'objet (*Object*). Il nous semble que ces quatre entités sont à la base de tout méta-modèle de processus. Nous y avons ajouté le concept de contrainte (*Constraint*), qui nous paraît nécessaire à la spécification de la structure de contrôle. Le processus peut contenir l'ensemble des éléments définis dans le méta-modèle. Les activités sont décomposables en sous-activités. Elles peuvent également définir des transitions entre ces sous-activités. La transition est l'unique mécanisme de spécification de dépendances entre activités. Elles peuvent être conditionnées par des contraintes. Enfin, les activités utilisent et produisent des objets. Leur réalisation est également confiée aux objets.

Les concepts ainsi définis peuvent aussi bien servir de base à des formalismes adaptés à la description de processus aussi complexes que les processus d'entreprise, que pour des méta-modèles plus modestes comme des organigrammes.

9.8. Conclusion

Dans ce chapitre, nous avons présenté notre vision d'un noyau de concepts pouvant être à la base de l'ensemble des méta-modèles de processus. Nous avons distingué quatre entités principales: les processus, les activités, les structures de contrôle et les objets. Nous avons évoqué le fait que le concept d'objet peut, dans de nombreux cas, être représenté dans un méta-modèle spécifique. Nous étudions cette question de façon plus précise dans le chapitre suivant qui traite de l'explicitation des relations entre produits et processus via les techniques de méta-modélisation.

Un noyau est composé de concepts qui semblent « universellement » utiles. Ceci signifie que pour toute définition de processus, les concepts proposés ont une utilisation directe. Un tel méta-modèle n'est bien sûr pas suffisant pour répondre à l'ensemble des besoins d'expression. Il fournit une base qui doit ensuite être étendue pour chaque domaine particulier. Ainsi, un méta-modèle dédié au processus de génie logiciel spécialisera le concept d'activité avec les notions de phases ou d'itérations alors qu'un méta-modèle pour l'ABC introduira des notions de coût. Nous avons vu que certains formalismes tels que PIF ou PSL proposait déjà une architecture modulaire, basée sur un noyau et des extensions. Une telle organisation peut être également définie dans des systèmes de méta-modélisation, grâce au mécanisme d'extension de paquetage. Une hiérarchie de méta-modèles permet ainsi de partager des mécanismes communs et de séparer les aspects disjoints.

10. Couplage des méta-modèles de processus avec d'autres types de méta-modèles

10.1. Introduction

Dans notre chapitre consacré à la définition d'un noyau commun à tous les formalismes de représentation de processus, nous avons introduit le concept d'objet. Il représente les artefacts utilisés et produits durant le processus. Ceux-ci sont contextuels au domaine d'application du processus. Ainsi, un processus de développement logiciel manipule des spécifications, modèles ou encore des portions de code tandis qu'un processus à base de services Web utilise les opérations fournies par ces services et échange des messages. Toutefois, quel que soit le domaine du processus, sa définition ne peut faire l'économie d'une description, au moins partielle, des artefacts manipulés. Comme on a pu le voir, de nombreux formalismes proposent donc des mécanismes, le plus souvent assez sommaires, pour la déclaration de ces artefacts. On définit alors, à l'aide d'un unique formalisme, les processus et les artefacts, ou plutôt un nombre limité d'aspects des artefacts.

Une telle solution conduit à un certain nombre de limites. Tout d'abord, les mêmes aspects d'un artefact peuvent être redéfinis à travers plusieurs modèles, ce qui peut poser des problèmes en terme de cohérence. De plus, la description sommaire d'une ressource peut être bloquante dans certains cas. C'est pourquoi le WfMC prévoit le couplage de WPDL avec un modèle de structure organisationnelle lorsque les mécanismes d'expression fournis par le formalisme de définition de workflow sont trop limités. Cette dernière solution, le couplage avec un modèle, voire des modèles, dédié à la description du domaine, peut être généralisée. Elle implique donc l'établissement de relations entre méta-modèles, permettant l'intégration du processus dans son domaine d'application par la définition de références.

En plus de l'intégration, de nombreux travaux ont conduit à la définition, souvent informelle, d'une autre forme de relation établissant des correspondances entre les structures des modèles. Cette relation a été particulièrement mise en évidence pour les modèles de processus et de produit. Ainsi, Reidar Conradi et al. déclarent : « *The product is evolved by activities, often driven by the product structure* » [14]. La structure du produit oriente les activités, c'est-à-dire la structure du processus. A l'inverse, August Scheer écrit : « *Not only are processes key in the manufacturing of products, but process forms also determine product types* » [84]. La forme du processus détermine le type du produit. Enfin, Ivar Jacobson dénie toute précédence à l'une ou à l'autre dimension en affirmant : « *product and process must be formed in a coordinated, integrated fashion* » [43]. Le produit et le processus doivent être conçus de façon coordonnée. Sans prendre part à une éventuelle

controverse sur la précédence de l'une ou l'autre dimension, nous voyons émerger des correspondances structurelles entre elles. Notre ambition n'est pas ici de décrire un ensemble de ces relations, mais d'étudier la façon dont elles peuvent être prises en compte à l'aide de méta-modèles explicites.

Nous commençons par nous intéresser à l'intégration, en illustrant notre discussion théorique par une expérimentation avec UML et PMM. Nous présentons ensuite diverses situations où l'existence d'une relation de correspondance structurelle a pu être mise en évidence, à défaut d'être précisément explicitée. Nous étudions les mécanismes pouvant permettre sa définition, avant de présenter quelques exemples de correspondances structurelles entre PMM et UML.

10.2. Intégration de méta-modèles

L'intégration de méta-modèles consiste à réunir des aspects disjoints et complémentaires d'un même système. L'utilisation de méta-modèles explicites permet de séparer puis de réunifier ces aspects. Ainsi, à l'OMG, un certain nombre de méta-modèles dédiés à des domaines d'intérêt différents ont été proposés ou sont en cours de définition. On peut par exemple citer UML, qui définit les objets constituant le système d'information, le méta-modèle de structure organisationnelle [64], ou encore le méta-modèle de workflow [69]. Il nous semble évident que des interactions existent entre ces méta-modèles. Cela est corroboré par des propositions comme le RAI [73] (Resource Assignment Interface) qui définit une APIs pour la gestion de l'affectation des ressources d'un workflow à l'exécution. L'affectation d'une ressource est contrainte par un certain nombre de règles, dont une partie au moins est issue du modèle de processus. Ce modèle est donc mis en relation (souvent de façon implicite) avec des modèles de ressources, pouvant être basés sur différents méta-modèles. L'intégration explicite des méta-modèles permet de définir précisément cette relation.

Intégrer des méta-modèles peut se faire très simplement au moyen des mécanismes d'extensions, à condition que les méta-modèles soient basés sur le même formalisme. Si l'on se place dans un environnement MOF, l'ensemble des méta-modèles doivent être basés sur le MOF. Il est ensuite très simple de définir un méta-modèle de mise en relation, qui étend l'ensemble des méta-modèles à intégrer en définissant de nouveaux concepts, de nouvelles relations et en ajoutant de nouvelles contraintes. Ce méta-modèle doit également être basé sur le MOF. Ainsi l'intégration d'un méta-modèle de processus et d'un méta-modèle de produit peut se faire de la façon présentée Figure 126.

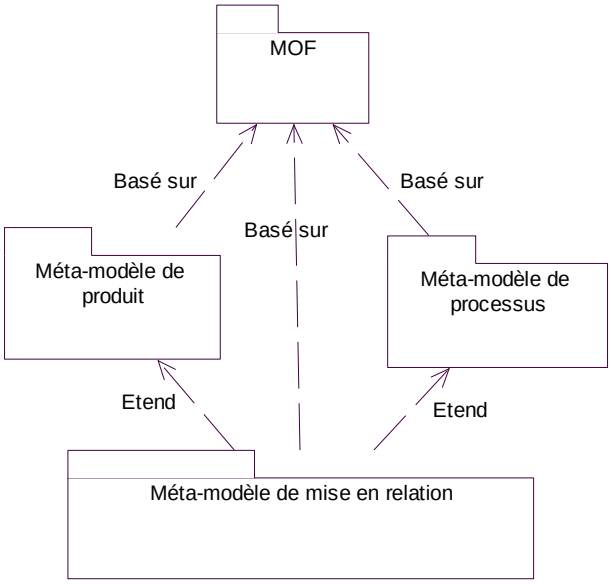


Figure 126. Intégration d'un méta-modèle de processus et d'un méta-modèle de produit

On peut constater que PMM et ARIS sont organisés de cette façon, c'est-à-dire que plusieurs méta-modèles décrivant des aspects partiels du système ont été définis. Ces méta-modèles sont ensuite mis en relation à l'aide d'un paquetage d'intégration. Ainsi, dans PMM, nous avons défini l'environnement d'exécution du processus comme un ensemble d'outils, de documents et de structures organisationnelles. Ces différentes dimensions sont ensuite intégrées au processus. L'intérêt d'une telle méthode est qu'il découple la définition du processus de son intégration dans un domaine particulier. Notre méta-modèle de définition de processus (*PMMProcessDefinition*) peut ainsi être réutilisé dans un autre contexte, par exemple pour spécifier des processus uniquement basés sur des invocations de services Web. Il suffit de définir une intégration, non plus avec les méta-modèles de définition d'outils, de documents et de structures organisationnelles, mais avec un méta-modèle de définition de services Web.

10.3. Intégration de PMM et d'UML

Lors de la présentation de nos réalisations, nous avons constaté un manque de richesse dans la définition de certains concepts de PMM. Par exemple, le système de typage des données du processus est assez sommaire. De même les composants invoqués ne sont que très partiellement décrits. Toutefois, pour conserver à notre méta-modèle une granularité assez limitée, nous pensons qu'il est plus intéressant d'étudier l'intégration de formalisme dédié à la description des parties qui nous font défaut, plutôt que de chercher à les ajouter. Dans cette partie, nous étudions donc son intégration possible avec UML. Pour cela, nous créons un méta-modèle de mise en relation qui étend à la fois PMM et UML (Figure 127).

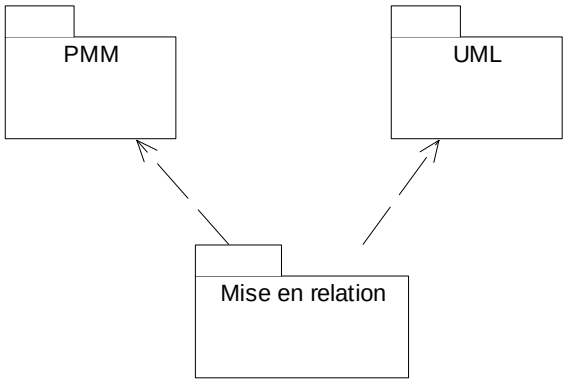


Figure 127. Un méta-modèle de mise en relation de PMM et d'UML

Une première relation qui peut être établie entre les deux méta-modèles concerne les données du processus. Le typage des données tel qu'il est fait dans le modeleur Sodifrance est relativement basique, puisqu'il repose sur une liste extensible de types de base. On ne dispose donc pas de la richesse d'expression d'UML pour la définition des classes. Pour cela, il serait nécessaire d'établir une relation entre le concept de *ProcessData* et celui de *Classifier*. L'ajout de cette relation n'est pas sans incidence. Le langage de contrainte utilisé pour les conditions de transition, les pré- et les post-conditions des activités et des tâches, est adapté à une typage fondé sur un ensemble de types simples, et non sur des classes. Dans l'optique d'une telle extension, son remplacement par un langage comme OCL devrait également être prévu.

Une seconde relation peut être définie entre les outils pouvant être invoquée au cours d'une tâche et l'opération UML que cet outil invoque. On peut alors créer un lien entre les deux éléments. De plus, une opération définit des paramètres. Lors de l'exécution d'une tâche, ces paramètres sont alimentés en entrée par des données du processus et mettent à jour ces dernières en sortie. Une relation peut donc être définie entre ces deux éléments. Cette relation pourrait être enrichie en observant qu'un paramètre peut être alimenté par une combinaison de données du processus, et réciproquement. Nous ne traitons pas ce cas ici. Nous avons donc défini le méta-modèle suivant (Figure 128).

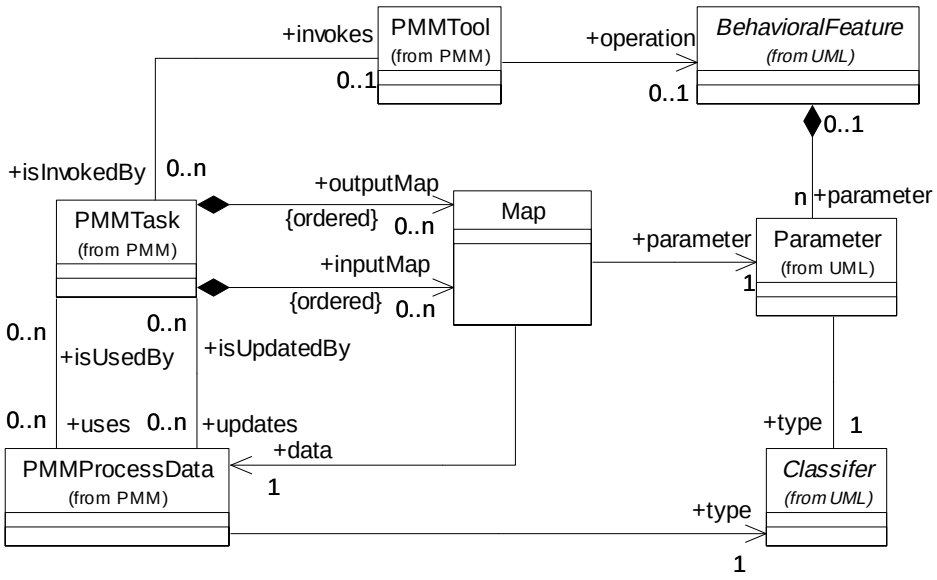


Figure 128. Mise en relation UML PMM

Une donnée (*PMMProcessData*) est typée par un *Classifier* UML. Un outil (*PMMTool*) peut être une opération ou une méthode (*BehavioralFeature*). Une tâche (*PMMTask*) peut nécessiter l'invocation d'une opération par l'intermédiaire d'un outil. Cette invocation nécessite la mise en correspondance (*Map*) des données du processus avec les paramètres de l'opération (*Parameter*). Une *Map* peut être en entrée ou en sortie. A ces concepts, il convient de rajouter un certain nombre de contraintes. Toutes *Map* en entrée doit mettre en relation une donnée utilisée par la tâche automatique avec un paramètre d'entrée de l'opération référencée par l'outil. De la même manière, toute *Map* en sortie doit mettre en relation une donnée mise à jour par la tâche avec un paramètre en sortie de l'opération. Cela peut se traduire en OCL de la façon suivante :

PMMTask

[1] Toute *Map* en entrée doit référencer un paramètre en entrée de l'opération référencée par l'outil associée à la tâche

```
self.inputMap->forAll(m : Map | self.invokes.operation.parameter->exists (Parameter p | p = m.parameter
and (m.parameter.kind="in" or m.parameter.kind="inout")))
```

[2] Toute *Map* en entrée doit référencer une donnée utilisée par la tâche

```
self.inputMap->forAll(m : Map | self.uses->exists (PMMProcessdata d | d = m.data))
```

[3] Toute *Map* en sortie doit référencer un paramètre en sortie de l'opération référencée par l'outil associée à la tâche

```
self.outputMap->forAll(m : Map | self.invokes.operation.parameter->exists (Parameter p | p = m.parameter
and (m.parameter.kind="out" or m.parameter.kind="inout")))
```

[4] Toute *Map* en sortie doit référencer une donnée mise à jour par la tâche

```
self.outputMap->forAll(m : Map | self.updates->exists (PMMProcessdata d | d = m.data))
```

Cette mise en relation permet donc de réutiliser le système de typage défini par UML, de définir les composants assistant la réalisation d'une tâche, ainsi que l'opération invoquée

et son paramétrage. Le processus peut alors être totalement intégré dans le système d'information décrit à l'aide de modèles UML.

10.4. Identifications de correspondances structurelles

De nombreuses méthodes d'analyse et de conception séparent la définition des structures de données de celle des traitements. Toutefois, dans de nombreux cas, les auteurs ont soupçonné l'existence d'une relation de dépendance entre ces deux dimensions. Du fait de manque de moyens d'expression à leur disposition, ces présomptions n'ont souvent été explicitées que sous la forme de patterns.

Par exemple, on peut citer la méthode SADT [51] (Structured Analysis and Design Technique), qui a été proposée par Douglas Ross à la société Softech en 1976-77. Cette méthode préconisait de décomposer le problème selon deux dimensions : la dimension activités et la dimension données. Les diagrammes d'activité étaient représentés par des actigrammes et les diagrammes de données par des datagrammes. Loin d'être indépendants, ces deux aspects devaient être confrontés en permanence afin de rester cohérents. En effet, la méthode originale recommandait de suivre l'une ou l'autre des voies et d'alterner l'analyse en changeant de voie et en vérifiant la similarité des démarches afin d'établir des correspondances et d'effectuer un contrôle croisé dans le but de rechercher d'éventuelles erreurs ou omissions

Autre exemple, la méthode JSD [42] (Jackson System Design), qui propose une façon de modéliser le flot d'entrée et le flot de sortie de données d'un programme en utilisant des arbres de structure (séquences de données, répétition, hiérarchie, alternatives, etc.). A partir de là, M. Jackson montre comment on peut se servir de ces informations pour dériver la structure de contrôle du programme (opérations et processus). Considérons par exemple un fichier qui débute par un enregistrement de type T1, suivi d'un nombre quelconque d'enregistrements de type T2 et se terminant par un seul enregistrement de type T3. Cette analyse s'exprime sur les données par un schéma identique à celui décrit en Figure 129. On déduit de ceci la structure de contrôle abstraite suivante.

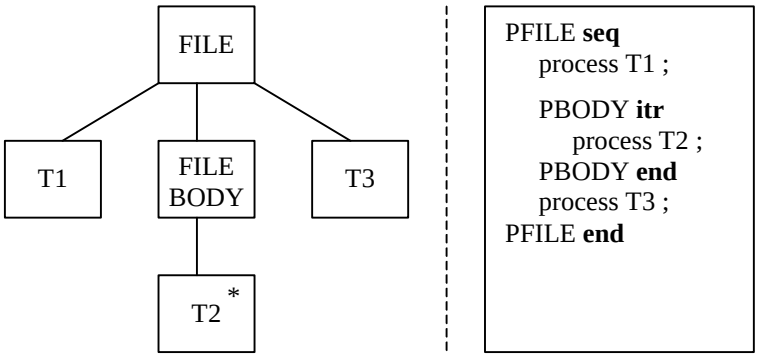


Figure 129. Dérivation d'un programme à partir d'un arbre de structure en JSD

Enfin, dans une approche similaire, N. Wirth a remarqué [105] la similarité de structure entre l'organisation des données et l'organisation du contrôle. Par exemple, l'exploration d'une structure de données récursive pourra se faire grâce à une procédure récursive, mais ceci n'est pas systématique. Cette similarité de structure apparaît plus clairement lorsque certaines notations sont utilisées (Figure 130), comme le langage Pascal par exemple :

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

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

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

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

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

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

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

Figure 130. Quelques exemples de similarité structure/algorithme

10.5. Correspondances structurelles et méta-modélisation

Dans les deux derniers exemples présentés précédemment, les auteurs ont établi des correspondances structurelles fortes entre les espaces produit et processus. A partir d'une structure de produit, on peut déduire une structure caractéristique de processus de traitement. Cela signifie donc que le processus peut être partiellement généré à partir du produit. On serait donc alors confronté, non plus à des références entre modèles, mais à des transformations de modèles.

Les travaux de Peter Loos et Thomas Allweyer ([53], [54]) viennent corroborer ce constat. Ils ont étudié les relations entre UML et les EPC (Event-driven Process Chains), qui sont une notation fréquemment utilisée pour représenter les processus. Ils permettent de capturer le flot d'activités et les événements qu'ils génèrent et peuvent être enrichis avec des informations telles que l'unité organisationnelle chargée de la réalisation d'une activité ou les données manipulées par une activité. Ils ont étudié un certain nombre de correspondances possibles entre les deux formalismes. Ils ont entre autre défini une relation d'affectation d'une fonction EPC, qui recouvre la notion d'activité, à une paquetage ou à une

classe UML. Cette relation est une relation d'intégration telle que nous l'avons défini précédemment. Ils ont également défini des règles de traduction permettant de produire un certain nombre de schémas UML, tel que le diagramme d'activités, à partir d'un modèle EPC.

La correspondance structurelle entre modèles de processus et de produit est donc différente de la relation d'intégration. Elle ne s'exprime pas à travers la mise en relation des méta-modèles de processus et de produit, mais par la spécification de règles de transformation. Les techniques de transformation de modèles vont donc nous permettre d'expliciter de façon plus précise les différents patterns identifiés. On peut tout de même imaginer de créer des relations entre les structures correspondantes identifiées dans chacun des modèles. Ce sont alors des relations de traçabilité, qui facilitent la mise au point incrémentale de chacun des modèles. Elles peuvent être déduites à partir des règles de transformation.

10.6. Correspondances structurelles entre PMM et UML

Nous avons pu établir précédemment quelques relations définissant l'intégration de PMM avec UML. Nous cherchons maintenant à établir quelques correspondances structurelles entre ces deux formalismes. Pour cela, nous prenons l'exemple d'un processus de maintenance simplifié. Nous avons commencé par le représenter dans le formalisme PMM à l'aide du modeleur de processus (Figure 131).

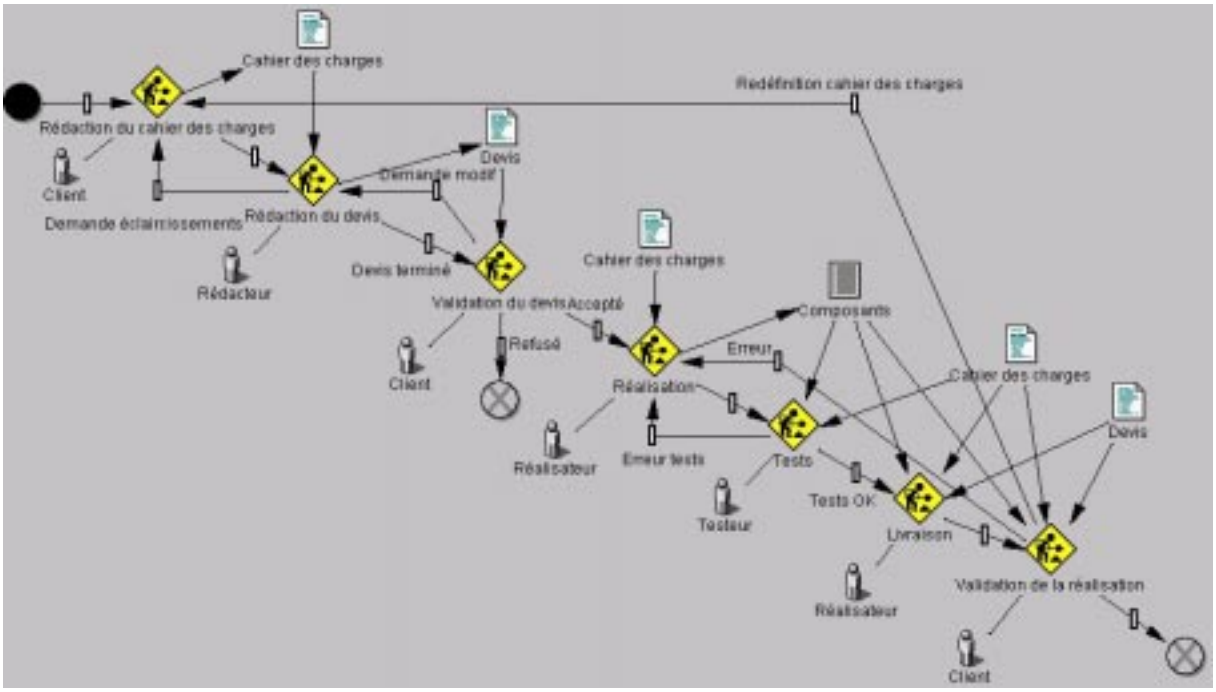


Figure 131. Description du processus avec le modeleur Sodifrance

Une première correspondance facilement identifiable est la représentation du processus précédent sous la forme d'un graphe d'activité (Figure 132). Une telle transformation n'amène toutefois pas énormément de bénéfices. Elle permet simplement de

visualiser le processus sous un autre formalisme, pas forcément aussi adapté puisque certaines informations ne peuvent être représentées (sauf à utiliser un profil). Il ne s'agit donc pas là d'une réelle correspondance structurelle entre deux méta-modèles dédiés à des domaines différents, mais plutôt d'une correspondance structurelle entre deux méta-modèles de représentation de processus.

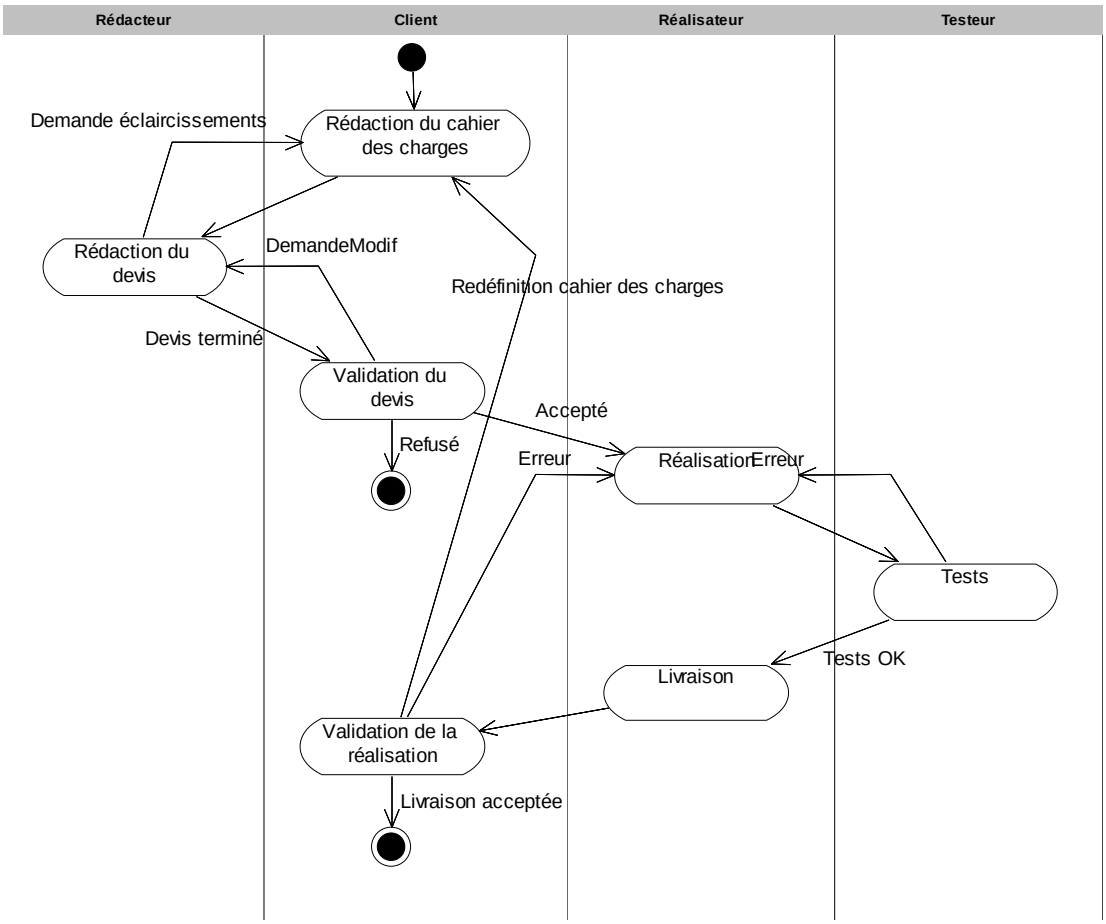


Figure 132. Description du processus avec un diagramme d'activité UML

Une transformation qui nous semble plus pertinente est la production d'un ensemble de cas d'utilisation à partir d'un processus. En effet, si l'on considère que chaque tâche du processus met en œuvre des composants logiciels du système d'information, on peut définir chacune d'entre elles comme un cas d'utilisation de ce système. Ainsi, le processus, les activités et les tâches vont produire des cas d'utilisation. De plus, la décomposition d'un processus ou d'une activité en sous-étapes peut être représentée par une inclusion de cas d'utilisation. Enfin, les rôles vont être transformés en acteurs, la réalisation d'une tâche manuelle par un rôle donnant lieu à une association entre l'acteur et le cas d'utilisation dans le modèle UML. Le processus présenté précédemment est transposé de la façon suivante dans une architecture de cas d'utilisation (Figure 133).

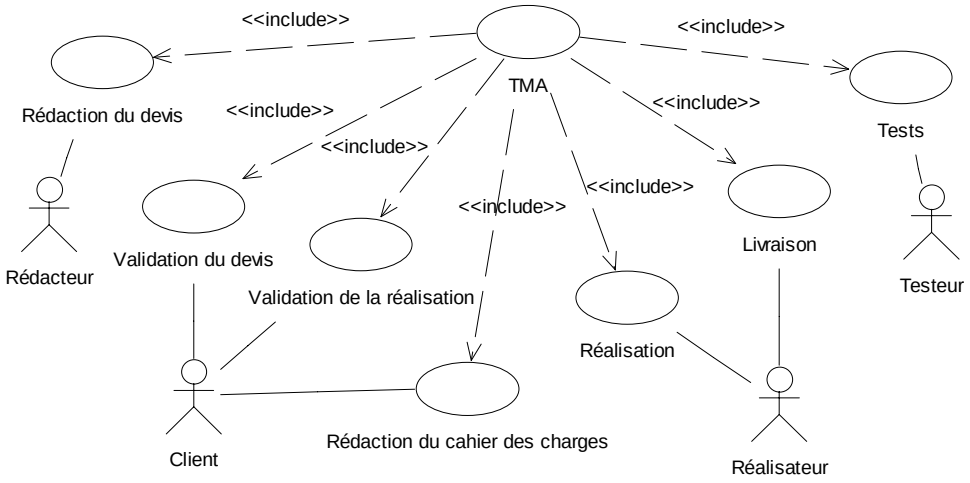


Figure 133. Définition d'une arborescence de cas d'utilisation UML à partir du processus

Ce pattern de correspondances peut être formalisé par l'écriture de règles de transformation de modèles. Ainsi, si on utilise Scriptor-T, on a la séquence de règles suivante (Figure 134).

```
// Tout processus génère un cas d'utilisation
PMMProcessPackage(p) name(p,n) -> UseCase(uc) name(uc,n) _coref(p,uc);

// Toute étape (activité, tâche) génère un cas d'utilisation
PMMActionStep(as) name(as,n) -> UseCase(uc) name(uc,n) _coref(as,uc);

// Tout rôle génère un acteur
PMMRole(r) name(r,n) -> Actor(a) name(a,n) _coref(r,a);

// La décomposition d'un processus en sous-étapes génère l'inclusion des cas d'utilisation correspondants
PMMProcessGraph(g) linktoSuperStep (g,linkSuper) process(linkSuper,p) _coref(p,ucp)
                        linktoContainedSteps(g,linkContained) step(linkContained,s) _coref(s,ucs) ->
        Include(inc) base(inc,ucp) addition(inc,ucs);

// La décomposition d'une activité en sous-étapes génère l'inclusion des cas d'utilisation correspondants
PMMActivityGraph(g) linktoSuperStep (g,linkSuper) activity(linkSuper,a) _coref(a,uca)
                        linktoContainedSteps(g,linkContained) step(linkContained,s) _coref(s,ucs) ->
        Include(inc) base(inc,uca) addition(inc,ucs);

// L'affectation d'un rôle à une tâche manuelle génère
// une association entre l'acteur et le cas d'utilisation correspondants
PMMManualTask(man) _coref(man,uc) linktoPerformerRole(man,link) role(link,r) _coref(r,a) ->
        Association(asso) AssociationEnd(endA) type(endA,a) association(endA,asso)
                        AssociationEnd(endUC) type(endUC,uc) association(endUC,asso);
```

Figure 134. Règles de production d'une architecture de cas d'utilisation à partir d'un processus PMM

10.7. Conclusion

Dans cette partie, nous avons présenté deux types de couplages entre les modèles de processus et d'autres méta-modèles. Nous avons tout d'abord défini la relation d'intégration entre méta-modèles, qui peut être spécifiée à l'aide de références entre leurs éléments respectifs. Nous avons également défini une relation de correspondance structurelle, qui s'exprime au travers de règles de transformation. Les techniques de méta-modélisation nous ont permis d'expliciter chacune de ces relations.

Cette explicitation n'est pas une fin en soi. En effet, la définition de ces relations ne définit pas pour autant la démarche à adopter lors de la conception d'un système. Dans quel ordre les modèles doivent-ils être définis ? Quels sont les patterns de transformation à adopter ? Comment maintenir la cohérence entre les différents modèles ? Autant de questions qui restent toujours ouvertes. Nos travaux n'y répondent pas de façon définitive. Toutefois, une spécification précise des relations entre méta-modèles constitue une première étape vers la mise au point d'une telle démarche, basée sur les méta-modèles. Dans ce chapitre, nous avons fourni quelques éléments permettant de mieux comprendre ces relations, et les mécanismes qui permettent de les représenter.

11. Vers des modèles de processus dynamiques

11.1. Introduction

Lors de notre présentation des techniques de méta-modélisation, nous avons défini une relation entre le système et le modèle : un modèle représente un système. Il existe deux types importants de systèmes : les systèmes statiques et les systèmes dynamiques. Un système statique demeure inchangé dans le temps. Poser une même question à un tel système à deux moments différents aboutit à une même réponse. A l'opposé, les systèmes dynamiques évoluent au cours du temps. La même question posée à un tel système à deux moments différents peut alors donner des résultats différents. La grande différence entre ces deux types de systèmes se situe donc, comme l'a remarqué Jackson [42], dans l'absence ou la présence de la dimension temporelle. Les systèmes statiques sont beaucoup plus rares dans la réalité que les systèmes dynamiques. Comme exemple de système statique, Jackson cite le résultat d'un recensement. Il donne en effet un instantané de la composition d'une population à un moment donné, un état figé sans passé ni futur. Les systèmes dynamiques sont bien plus courants. Par exemple la démographie d'un pays ou le processus de développement d'un logiciel sont des systèmes dynamiques.

Tout comme un système peut être statique ou dynamique, les modèles qui le représentent peuvent également être statiques ou dynamiques. Un modèle statique est un modèle qui n'évolue pas, alors qu'un modèle dynamique suit les évolutions (ou, tout du moins une partie d'entre elles) qui affectent le système qu'il représente. Un modèle statique est donc particulièrement adapté à la représentation de systèmes statiques et, parallèlement, un modèle dynamique l'est pour la représentation des systèmes dynamiques. La représentation d'un système statique par un système dynamique ne semble représenter aucun intérêt. Par contre, il n'est pas rare de représenter un système dynamique par un modèle statique.

Table 5. Cas possibles de représentation d'un système par un modèle

Peut représenter	Modèle statique	Modèle dynamique
Système statique	√	
Système dynamique	√	√

Dans la section présentant les techniques de méta-modélisation, nous avons défini que le méta-modèle peut être vu comme un filtre entre le système et le modèle. Cela est aussi bien vrai pour les systèmes statiques que pour les systèmes dynamiques (Figure 135). La manière dont le système est représenté, et plus particulièrement la prise en compte des aspects dynamiques du système, est donc uniquement contrainte par le méta-modèle. Selon les concepts et la sémantique que ce dernier introduit, ces aspects pourront être éludés ou présents.

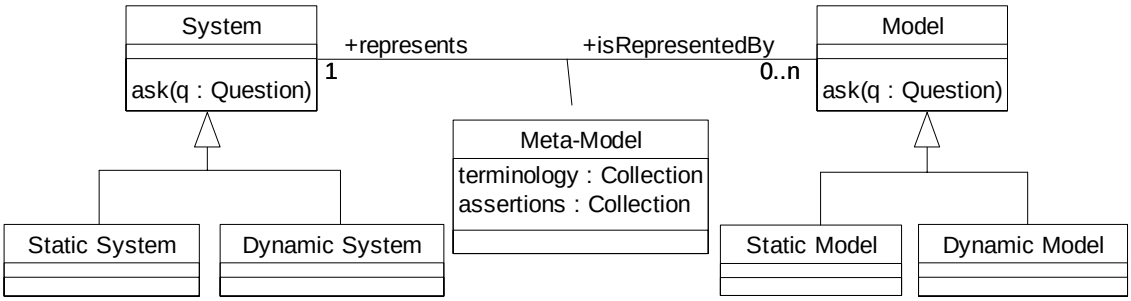


Figure 135. Systèmes, modèles et méta-modèles

Notre proposition est donc de définir les aspects dynamiques au même titre que les aspects statiques dans les méta-modèles de processus. Pour étayer notre propos, nous présentons deux expérimentations. La première définit un méta-modèle de réseaux de Petri intégrant ces deux aspects. La seconde réunit dans un même formalisme deux propositions, le méta-modèle de définition de processus du WfMC et une spécification de l'OMG relative aux interfaces des moteurs de workflow. Nous explicitons ensuite notre proposition et nous la mettons en perspective avec des travaux similaires.

11.2. Expérimentation avec les réseaux de Petri

Les réseaux de Petri sont un formalisme bien connu utilisé pour l'étude des communications entre des processus parallèles. Un réseau de Petri classique est un ensemble de places et de transitions connectées par des arcs orientés. Un arc va d'une place vers une transition ou d'une transition vers une place. Le réseau de Petri suivant (Figure 136) est constitué de quatre places (P1, P2, P3, P4) et de deux transitions (T1 et T2).

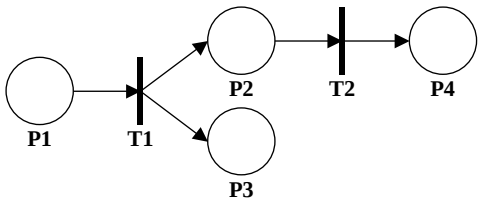


Figure 136. Un réseau de Petri

Le méta-modèle suivant (Figure 137) réunit l'ensemble de ces concepts.

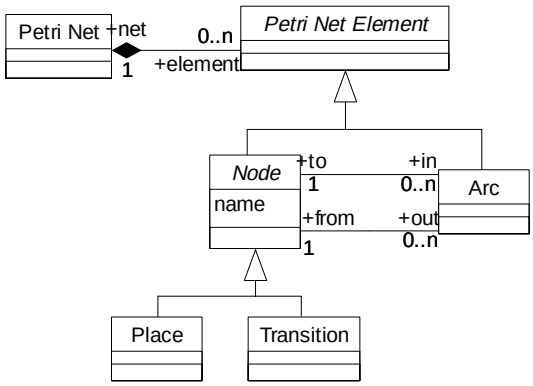


Figure 137. Méta-modèle d'un réseau de Petri

Un réseau de Petri est un ensemble d'*Arcs* et de *Nodes*. Un *Node* peut être soit une *Place* soit une *Transition*. La contrainte interdisant la création d'un arc entre deux places ou entre deux transitions peut être spécifiée en OCL de la manière suivante :

```
context Arc
    inv: self.to.oclIsTypeOf(Transition) and self.from.oclIsTypeOf(Place) or
         self.to.oclIsTypeOf(Place) and self.from.oclIsTypeOf(Transition)
```

Le réseau de Petri présenté Figure 136 peut donner lieu à un nombre infini d'exécution. Toutefois, les places, transitions et arcs ne seront pas modifiés par ces exécutions. Ils constituent la partie statique du modèle. Le méta-modèle Figure 137 définit les concepts de cette partie statique. Ainsi, on peut représenter des réseaux de Petri et raisonner sur ces modèles. Le graphe peut être simplifié par l'application de règles de réduction, des points de contention peuvent être découverts et éliminés, etc.

Ces concepts vont définir des éléments communs à un certain nombre de situations particulières, mais ils ne permettent pas de définir précisément l'ensemble de la situation. Une situation particulière d'un réseau de Petri au cours d'une exécution est constituée par un marquage, c'est-à-dire un ensemble de jetons positionnés sur un ensemble de places.

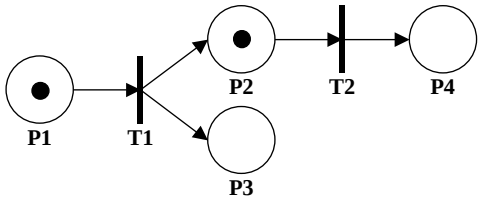


Figure 138. Une situation particulière lors d'une exécution d'un réseau de Petri

Si l'on souhaite pouvoir représenter une situation, il nous faut enrichir le méta-modèle avec les concepts de marquage (*Marking*) et de jeton (*Token*) (Figure 139).

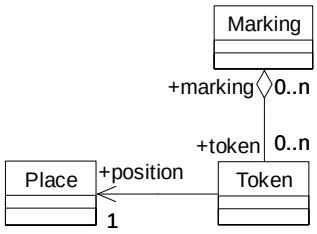


Figure 139. Définition du marquage

Une exécution d'un réseau de Petri (du moins dans le cas standard) est une succession de marquages. Entre chaque marquage une transition a été sélectionnée puis mise à feu. Cela donne lieu à la suppression d'un jeton de chacune des places précédant la transition, et à la création d'un jeton à chacune des places suivant la transition (Figure 140).

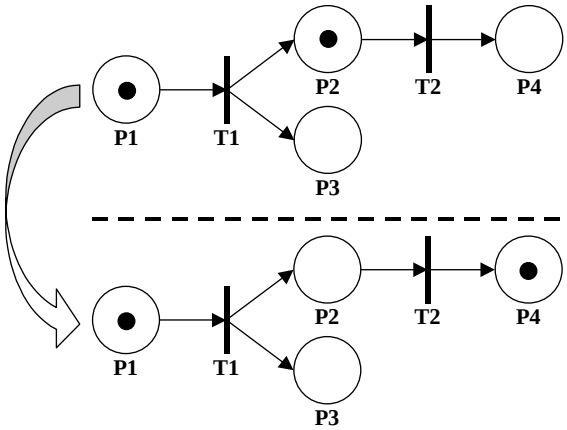


Figure 140. Deux marquages successifs avec mise à feu de la transition T2

L'exécution d'un réseau de Petri peut alors être vue comme un ensemble de mouvements qui vont mener d'un marquage à l'autre. Un mouvement consiste à mettre à feu une transition. Parmi tous les marquages, on peut distinguer le marquage initial et le marquage courant. Le méta-modèle présenté Figure 141 introduit l'ensemble de ces concepts.

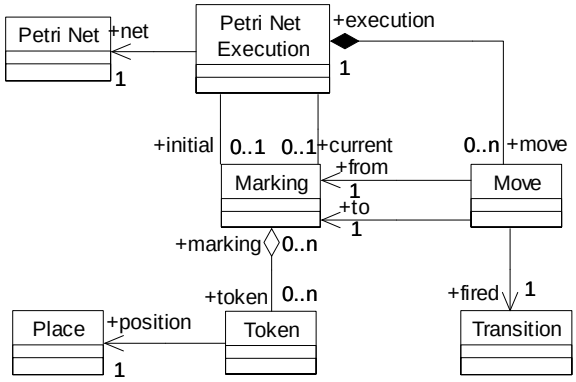


Figure 141. Méta-modèle de l'exécution d'un réseau de Petri

On peut alors définir le méta-modèle des réseaux de Petri comme l'agrégation de deux méta-modèles, le méta-modèle de définition, la partie statique, et le méta-modèle d'exécution, la partie dynamique (Figure 142). Cette dernière est fortement dépendante de la définition, puisqu'on a pu établir des relations entre l'exécution et le réseau, le jeton et la place, le mouvement et la transition mise à feu.

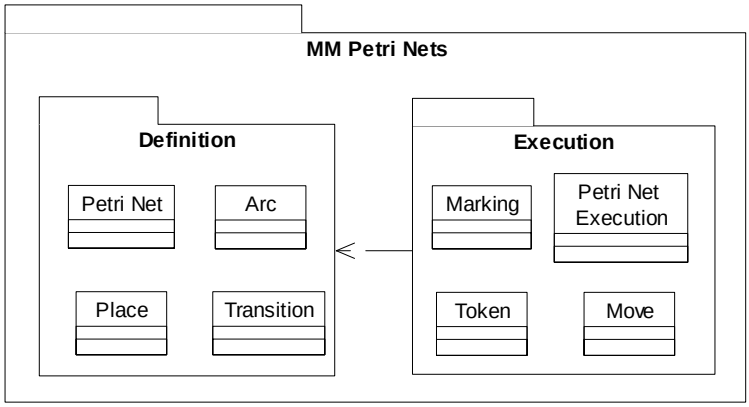


Figure 142. Identification de deux parties dans un réseau de Petri

Un des avantages de cette séparation entre aspects statiques et aspects dynamiques est que l'on peut explicitement associer divers modèles d'exécution à un même formalisme de définition. Chacune de ces interprétations peut être représentée comme un paquetage. Suivant qu'elles sont des variations ou qu'elles intègrent des différences majeures (par exemple si on ne matérialise plus l'exécution d'un réseau de Petri par des jetons mais par des états), les paquetages correspondants s'étendent, et les concepts définis à un niveau peuvent être réutilisés (voir Figure 143).

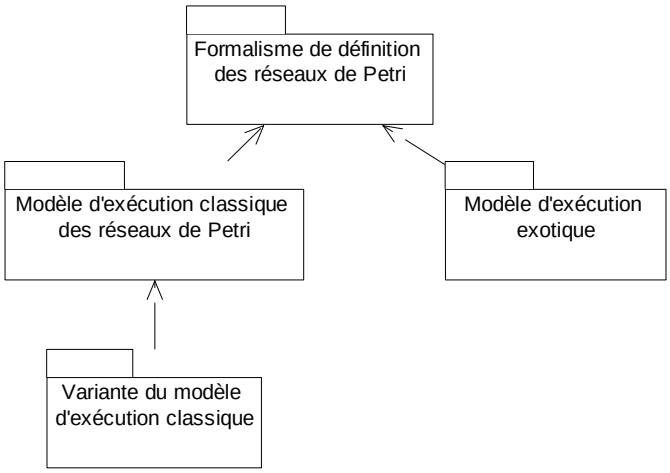


Figure 143. Architecture des formalismes de définition et des modèles d'exécution des réseaux de Petri

L'explicitation de ces modèles d'exécution permet d'identifier aisément leurs variations.

11.3. Dans le domaine du workflow

Dans la section dédiée à l'étude de différents formalismes de représentation de processus, nous avons présenté le méta-modèle de définition de processus proposé par le WfMC. Ce méta-modèle permet de décrire des modèles de processus. Il ne définit donc que la partie statique du méta-modèle, c'est-à-dire les concepts communs entre plusieurs exécutions et qui ne subissent aucune transformation lors de ces exécutions. Tous les concepts susceptibles d'évolution sont donc absents de cette spécification, bien qu'il y soit reconnu que la définition d'un certain nombre de principes de base et de la sémantique soit nécessaire (« *since a build time representation of the process definition will subsequently be used to support enactment it is necessary to define a number of basic principles and semantics of the execution time environment* ») [106].

Ces principes de base sont justement au cœur de la spécification de l'OMG sur le Workflow Management. Ce modèle UML propose un ensemble de classes standards pour la manipulation des processus lors de leur exécution (Figure 144). Cette spécification propose une interface standard pour les moteurs de workflow. Elle ne définit pas de façon précise la manière dont chacune des opérations pouvant être invoquée depuis l'extérieur doit être implémentée, pour le contrôle de l'exécution, le suivi et l'interopérabilité entre moteurs de workflow. Toute implémentation de cette spécification est guidée par des choix architecturaux ou liés à l'infrastructure, mais aussi fortement par le formalisme de définition des processus. Dans cette partie, nous étudions comment une implémentation de cette spécification pourrait être associée au méta-modèle de définition de processus du WfMC.

Dans le modèle d'exécution spécifié par l'OMG, un processus (*WfProcess*) est initié par un demandeur (*WfRequester*). Ce demandeur est associé au processus lors de la création de ce dernier. Il est notifié de tout changement d'état au sein du processus. Un demandeur peut être soit une application externe, soit une activité (*WfActivity*), qui est alors dans ce cas un sous-processus. Le processus est créé par un manager (*WfProcessMgr*), qui représente une définition de processus particulière. Il génère et il contient les instances de processus. L'exécution du processus est démarrée en invoquant l'opération *start*. Les activités de départ sont alors démarrées. Une activité peut être implémentée par un sous processus, et dans ce cas l'activité doit attendre la fin de ce sous-processus pour pouvoir terminer. Elle peut également être réalisée par une application externe, et dans ce cas l'application signale explicitement la fin de l'activité grâce à la méthode *complete*. Une activité ne peut être explicitement démarrée depuis une application externe. Le moment de son activation est défini dans le modèle du processus. De même, un processus ne peut être marqué comme complet par l'invocation d'une méthode. Un processus n'est complet que lorsqu'il n'y a plus aucune activité en cours. Par contre, à tout moment, une activité et un processus peuvent être suspendus (*suspend*), repris (*resume*) ou stoppés (*abort*, *terminate*). Leurs états évoluent selon le modèle défini dans le formalisme de représentation du WfMC. Activités et processus peuvent également générer des résultats (*result*). Les résultats d'une activité peuvent être utilisés, une fois que celle-ci est terminée, afin d'orienter le flux de contrôle. Ils peuvent également être intégrés aux résultats du processus. Les activités peuvent nécessiter des ressources (*WfResource*), c'est-à-dire des composants qui accepteront des affectations temporaires (*WfAssignment*). Enfin, un certain nombre d'événements (*WfEventAudit*) sont mémorisés. Il s'agit de la création de processus (*WfCreateProcessEventAudit*), du changement d'état d'un processus ou d'une activité (*WfStateEventAudit*), du changement de

valeur d'une donnée (*WfDataEventAudit*) ou encore d'une modification dans les affectations de ressources (*WfAssignmentEventAudit*).

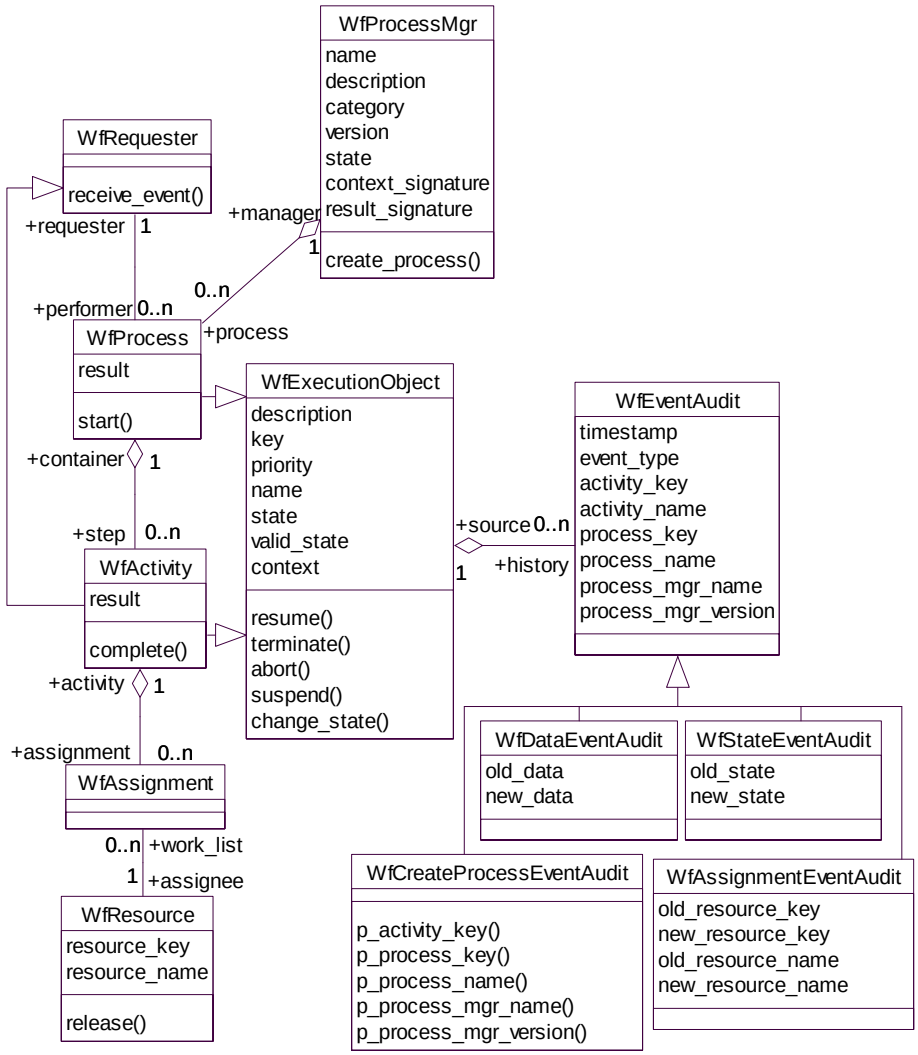


Figure 144. Le modèle d'exécution de l'OMG

Une brève étude du méta-modèle du WfMC et du modèle d'exécution de l'OMG permet d'établir un certain nombre de correspondances comme le montre le tableau suivant :

Table 6. Correspondances entre concepts du méta-modèle de la WfMC et du modèle de l'OMG

WfMC	OMG
WorkflowProcessDefinition	WfProcess
WorkflowProcessActivity	WfActivity
WorkflowParticipantSpecification	WfResource

WorkflowRelevantData	result

Ces deux modèles peuvent donc être fusionnés au sein d'un seul et unique modèle de la façon suivante (Figure 145):

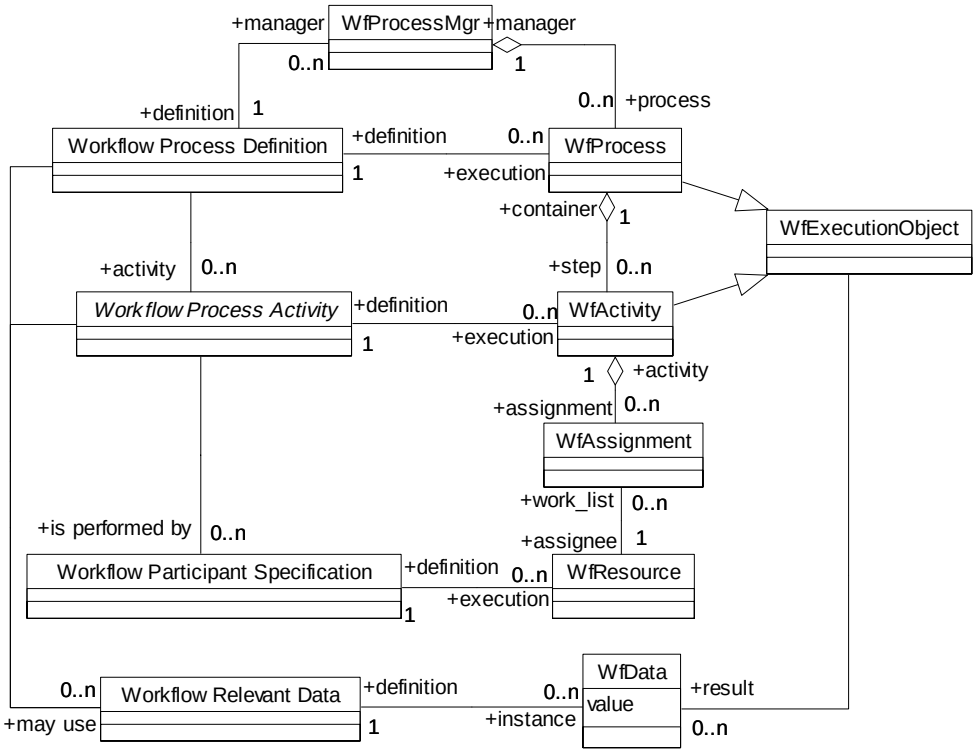


Figure 145. Couplage des concepts statiques et dynamiques

On a ainsi créé une association entre les éléments du modèle de définition et ceux du modèle d'exécution. L'association entre *Workflow Process Definition* et *WfProcess* indique que le *WfProcess* est une exécution du *Workflow Process Definition*. Il en va de même pour l'association entre *Workflow Process Activity* et *WfActivity*. Enfin, nous avons introduit la notion de données d'exécution (*WfData*) , qui représente les résultats des processus et des activités. Elle peut être considérée comme une instance de la *Workflow Relevant Data*. Cette dernière définit le type de la *WfData*, qui, elle, ne contient qu'une valeur. Nous avons ainsi pu établir un lien entre de nombreuses entités du méta-modèle de définition et du modèle d'exécution. A partir de là, on peut définir qu'une activité (*WfProcess*) ne peut démarrer un sous processus que si sa définition est elle-même un sous processus (*SubProcessDefinition*, voir la présentation du méta-modèle du WfMC). On peut donc ajouter la contrainte suivante à notre modèle :

```
context Activity
    inv: not self.definition.oclIsTypeOf(SubProcessDefinition)
        implies self.performer->isEmpty
```

Nous avons aussi pu identifier un autre type de correspondances entre les deux modèles. Il s'agit de la relation entre le statut du processus (*Worklow Process Definition*) et l'état du manager (*WfProcessMgr*). La WfMC a défini les différents statuts pour une définition de processus : *UNDER_REVISION*, *RELEASED*, *UNDER_TEST*. De son côté, l'OMG a défini les états suivants pour le manager : *enabled*, *disabled*. Il nous semble qu'il existe un lien entre ces deux diagrammes d'état. Ainsi, le passage d'une définition de processus dans l'état *UNDER_REVISION* devrait avoir pour résultat de faire passer l'ensemble de ses managers dans l'état *disabled*.

A l'origine, le méta-modèle de définition de processus et le modèle d'exécution étaient complètement séparés. Le premier était basé sur le MOF et le second sur UML (Figure 146). Il n'y avait aucune relation définie entre ces formalismes. Cette séparation se retrouvait encore entre les modèle de processus et les exécutions.

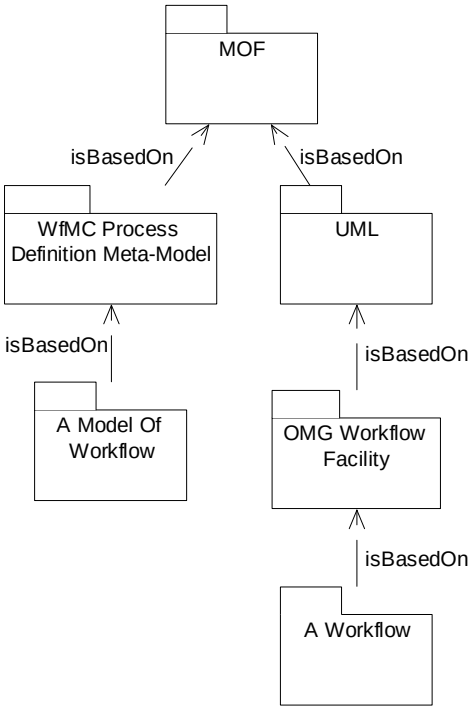


Figure 146. Séparation des modèles de définition et d'exécution

Nous avons réuni ces deux propositions au sein d'un même formalisme basé sur le MOF. Les modèles de processus se situent alors au même niveau de modélisation que leurs exécutions. Nous sommes donc passés d'une architecture de modélisation en quatre couches à une organisation en trois couches (Figure 147).

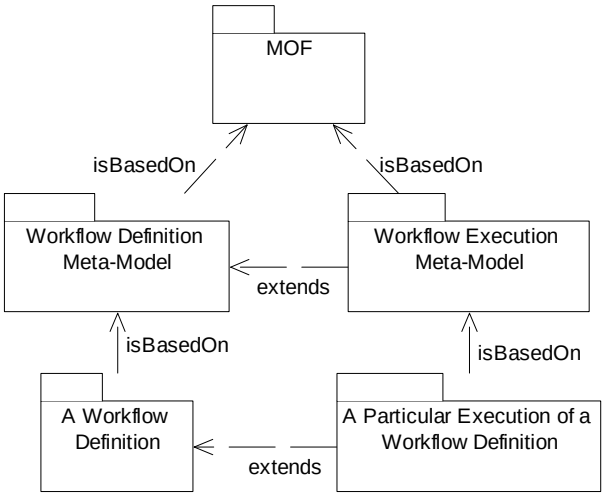


Figure 147. Couplage des modèles de définition et d'exécution

11.4. Généralisation

L'architecture de modélisation définie à l'OMG à la suite des travaux sur le MOF est basée sur quatre couches distinctes. Il y a la couche du méta-méta-modèle, celles des méta-modèles, celles des modèles et enfin la couche des instances. Si on prend l'exemple des processus, on a alors :

- Le méta-méta-modèle, tel que le MOF, unique et réflexif
- Un méta-modèle de processus quelconque, introduisant par exemple les concepts de processus et d'activités
- Des modèles de processus, comme un processus d'achat ou un processus de développement logiciel
- En enfin, au plus bas niveau, les instances de ces processus.

C'est la structure typique telle que la définit Schulze [86] dans ces travaux préliminaires à l'intégration de concepts de définition de workflow au sein de l'OMA.

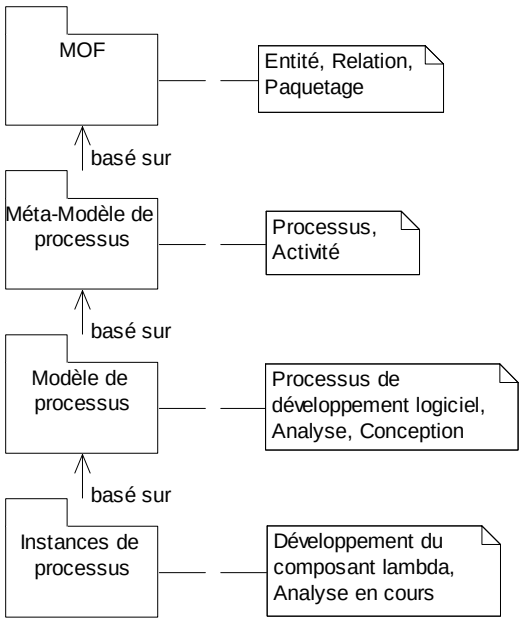


Figure 148. Les quatre niveaux traditionnels appliqués aux processus

Le problème est que cette architecture n'est pas utilisée dans les faits. Il suffit de se reporter à la Figure 146 pour constater que l'organisation est différente. Dans ce contexte l'instance de processus *Développement du composant lambda* n'est pas une instance du modèle de processus *Processus de développement logiciel*, mais de la classe UML *WfProcess*. L'exécution d'un processus est bien un concept au même titre que la définition de processus. Elle définit ses propres opérations et ses propres propriétés, qui sont différentes de celles de la définition. L'architecture à quatre niveaux omet cet aspect important en réduisant l'exécution de processus à l'instanciation d'une définition de processus. L'exemple des réseaux de Petri est alors très significatif, puisque les structures de données utilisées pour définir le graphe et celles utilisée pour représenter l'exécution sont alors très différentes.

L'architecture présentée Figure 146 permet de bien distinguer les deux aspects du processus, celui de la définition et celui de l'exécution. Toutefois, ces deux aspects étant exprimés à l'aide de formalismes différents, un méta-modèle basé sur le MOF et un modèle UML, ils sont difficilement réconciliables. En particulier, on ne peut pas établir de relations entre des concepts définis dans chacun des aspects, ni spécifier de contraintes permettant de conditionner le modèle de l'exécution du processus en fonction du modèle de définition.

Notre proposition, présentée Figure 147, consiste donc à décrire les aspects statiques et dynamiques des processus dans un même formalisme. On peut ainsi les coupler et spécifier leurs interdépendances. C'est ce qu'ont montré nos deux expérimentations. Nous avons identifié des concepts dédiés à la description générique d'un processus, et d'autres dédiés à la représentation d'une exécution particulière de ce processus. Les premiers forment la partie statique de méta-modèle. Un modèle basé sur ces concepts définit les règles contraignant l'exécution du processus. Il ne prend pas en compte la dimension temporelle. Les seconds forment la partie dynamique du modèle, celle qui évolue avec le temps.

Loin d'être indépendantes, ces deux parties sont fortement couplées. Ce couplage n'est pas forcément bilatéral. En effet, les concepts de la partie dynamique dépendent de ceux de la partie statique. Ainsi, dans l'exemple des réseaux de Petri, le jeton est positionné à une place. Par contre, les concepts de place et de transition ne sont en aucun cas dépendants de celui de jeton. La définition d'un processus peut donc se faire indépendamment de toute préoccupation d'exécution.

Cette introduction du concept d'instance de processus au même niveau que celui de processus a pu être observée dans un certain nombre de formalismes que nous avons présentés précédemment. Ainsi, PSL définit-il les concepts d'activité (*activity*) et d'occurrence d'activité (*occurrence_activity*). Une occurrence d'activité est l'occurrence d'une et une seule activité. Elle est caractérisée par des instants de début et de fin ainsi que par un ensemble d'objets qui prennent part à cette activité sur une certaine durée. De la même façon, CPR a distingué deux types de plans : les plans de conception et les plans d'exécution. Un plan d'exécution correspond à un plan de conception. Ces deux types de plans ont des contenus très similaires. Mais, alors qu'un plan de conception prévoit une consommation de ressources ou définit des critères de calcul de réalisation des objectifs, le plan d'exécution indique une consommation effective de ressources et les taux de réalisation affectés à chacun des objectifs.

On peut donc constater que cette préoccupation de la définition intégrée des concepts statiques et dynamiques est partagée. Les travaux sur Noesis [21] vont également dans ce sens. L'approche Noesis est destinée à la représentation des caractéristiques comportementales. Parmi les notions de base de Noesis, on trouve également la séparation des méta-modèles en deux couches :

- La couche « status-independent », qui capture la structure statique
- La couche « status », qui capture la dynamique.

A ces couches s'ajoutent les notions de transformation. Une transformation T0 permet de déterminer, à partir de la structure statique, l'état initial de la couche dynamique. Sur cette couche dynamique peuvent s'appliquer un certain nombre de transformations, qui définissent en fait la manière dont le système se comporte. Cet aspect supplémentaire est décrit dans le chapitre suivant.

11.5. Conclusion

Dans ce chapitre, nous avons montré que les méta-modèles de processus peuvent être découpés en deux parties distinctes mais dépendantes, la partie de définition du processus et la partie spécifiant le modèle d'exécution. Nous avons également montré qu'il ne faut pas confondre les relations entre ces deux parties avec des relations d'instanciation, comme c'est fréquemment le cas. Ces travaux peuvent être comparés avec ceux poursuivis actuellement par Nicolas Ploquin et Jean Bézivin sur l'extraction de modèles d'exécution de programmes C# [8]. Les questions qui se posent concernent alors le formalisme utilisé pour représenter

l'exécution du programme ainsi que les relations existant entre ce modèle et le modèle du programme C#.

12. Vers des modèles de processus exécutables

12.1. Introduction

Dans la section précédente, nous avons présenté notre vision des modèles dynamiques. Leur principale caractéristique est leur évolution dans le temps. Cette évolution est cohérente avec celle du système représenté. Ce dernier peut évoluer de façon totalement aléatoire ou en fonction de règles bien précises. Dans ce dernier cas, il serait intéressant de définir ces règles au sein du modèle. Ainsi, on pourrait déduire, à partir d'une description du système à un moment t, l'état du système à un moment t+1 en fonction de différents facteurs d'évolution. L'intérêt d'un tel mécanisme est que l'on passe de questions de type « what is ? » portant sur la structure du modèle, à des questions de type « what if ? » qui s'intéressent à son évolution.

Dans ce chapitre, nous nous intéressons à la manière dont la méta-modélisation peut fournir des mécanismes permettant de rendre des modèles exécutables. Nous commençons par une discussion sur la définition de la sémantique des méta-modèles. Nous revenons ensuite sur les expérimentations que nous avons présentées dans le chapitre précédent. Nous montrons comment nous avons pu spécifier l'exécution d'un réseau de Petri dans l'environnement sNets. Nous présentons ensuite notre proposition, l'intégration au niveau du MOF d'un formalisme permettant de définir une sémantique basée sur les actions. Cette proposition s'inspire des deux formalismes que nous avons présentés en première partie : Action Semantics for Programming Language (AS_PL) et Action Semantics for UML (AS_UML). Nous finissons en illustrant notre proposition par un exemple.

12.2. Où est la sémantique des méta-modèles ?

Steve Cook identifie trois aspects principaux dans un langage de modélisation : la syntaxe concrète, la syntaxe abstraite et la sémantique [16]. La syntaxe concrète peut être définie sous la forme d'une grammaire, d'un dialecte XML ou encore d'un ensemble de formes graphiques. La syntaxe abstraite définit les concepts de base du langage. Enfin, la sémantique définit la signification des modèles. Ces trois aspects sont reliés par des liens de correspondance. On peut ainsi établir une correspondance entre syntaxe concrète et syntaxe abstraite, ainsi qu'une correspondance entre syntaxe abstraite et sémantique (Figure 149).

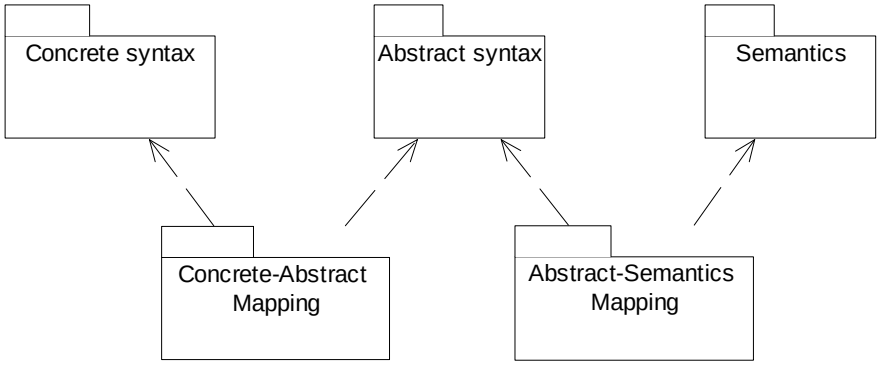


Figure 149. Les trois aspects des langages de modélisation (extrait de la présentation de Steve Cook à OCM 2002)

Les spécifications de l'OMG ne prennent en compte qu'une partie de ces aspects. Le MOF permet de définir une grammaire abstraite. XMI définit les règles de production d'une syntaxe concrète sous la forme d'un dialecte XML à partir d'un méta-modèle. D'autres travaux sont en cours à l'OMG pour définir d'autres média de syntaxe concrète. On peut citer le RFP sur le formalisme pour l'échange de diagrammes UML [66] ou encore de travaux sur la définition d'une notation textuelle lisible pour EDOC [19]. Nous avons également mené quelques investigations dans ce domaine durant notre thèse, en encadrant un projet d'étudiants d'IUP 3ème année sur la production d'environnements de modélisation graphique à partir de méta-modèles MOF. Ce travail a consisté à définir un formalisme graphique basé sur le MOF et son couplage avec des méta-modèles MOF. Nous avons ainsi pu exprimer explicitement la notation graphique d'un méta-modèle, et produire un environnement de travail correspondant. Cette dernière étape a été réalisée à l'aide de Scriptor-G. A partir de méta-modèles enrichis avec des informations graphiques, des gabarits ont été générés pour le logiciel Microsoft Visio.

Par contre, une des lacunes de l'architecture existante tient au manque de précision de la sémantique des méta-modèles. Ainsi, si la spécification d'UML est accompagnée d'une DTD XMI, la sémantique du méta-modèle est uniquement définie de manière textuelle.

Où est la sémantique d'exécution d'un méta-modèle de processus ? Où se trouvent les règles qui vont nous permettre d'interpréter un modèle ? Il faut bien reconnaître qu'aujourd'hui elles sont totalement implicites au niveau du méta-modèle. Elles peuvent être écrites en langage naturel dans la documentation accompagnant le méta-modèle. La sémantique d'exécution se trouve dans ces documents, puis dans les intentions des utilisateurs humains, puis dans les composants logiciels qui permettent de manipuler ces modèles. Un moteur de workflow attribue une sémantique aux modèles dont il contrôle l'exécution. Dans le cadre de nos réalisations industrielles, nous n'avons pas spécifié de sémantique d'exécution précise à PMM. Par contre, comme nous avons transformé les modèles basés sur PMM dans des formalismes exécutables, on pourrait dire que la sémantique de PMM est définie par sa transformation. Ainsi, la production d'un workflow pour FlowMind à partir d'un modèle PMM donne à ce dernier une sémantique d'exécution particulière. Toutefois, cette solution peut introduire un certain nombre d'ambiguïtés. Tout d'abord, différents formalismes cibles, avec chacun leurs particularités, peuvent être utilisés.

D'autre part, nous avons pu définir plusieurs transformations différentes entre PMM et FlowMind. Laquelle définit la sémantique de PMM ? C'est pourquoi la sémantique d'exécution d'un méta-modèle de processus doit pouvoir être définie au sein même de ce méta-modèle.

Cette lacune dans la définition des méta-modèles pose un certain nombre de problèmes. Tout d'abord, étant donné que la sémantique est exprimée sous une forme imprécise, différentes interprétations sont possibles. Et ainsi, deux moteurs de workflow basés sur le même méta-modèle de définition peuvent présenter des comportements différents. De plus, le lien entre le méta-modèle et les composants logiciels qui l'utilisent n'est pas direct. Il y a nécessité d'une intervention humaine, qui peut s'avérer parfois relativement coûteuse. Cela n'est pas gênant dans le cas où les méta-modèles ne sont pas sujets à modifications. Comme l'indique Michael Uschold [97] à propos du Web sémantique, plus il y a de consensus et moins l'on a besoin de formaliser la sémantique. Il prend par la suite l'exemple du tag « h2 » en HTML qui est interprété de la même façon par l'ensemble des navigateurs Web. Or, un des principaux avantages des méta-modèles est leur extensibilité. UML, autour duquel s'est formé un consensus, est régulièrement adapté à des besoins particuliers par le biais des mécanismes d'extensions (profils, tagged values, stéréotypes). Comme le remarque Giese [30], ni la sémantique d'UML, ni celle de ces différents profils, n'étant précisément spécifiée, il y a un risque d'incohérence.

12.3. Retour sur les réseaux de Petri

Dans le méta-modèle de réseaux de Petri présenté précédemment, nous avons pu mettre en évidence des aspects dynamiques et les relier aux aspects statiques. Nous avons en particulier introduit le concept de mouvement (*Move*), correspondant à la mise à feu d'une transition permettant la création d'un nouveau marquage. Toutefois, nous n'avons pas spécifié la manière dont se fait le passage d'un marquage à l'autre. On peut bien sûr utiliser des contraintes OCL pour contrôler si le marquage résultat d'un mouvement est valide. Mais la séquence d'opérations à effectuer n'est pas précisée. Cette séquence pourrait correspondre à la liste d'opérations suivantes :

- Sélectionner une transition parmi les transitions éligibles du marquage courant.
- Créer un mouvement et le relier à la transition sélectionnée ainsi qu'au marquage initial.
- Créer un nouveau marquage et lui ajouter les jetons du marquage courant sauf un à chacune des places précédant la transition sélectionnée.
- Créer un jeton à chacune des places suivant la transition sélectionnée et les ajouter au nouveau marquage.
- Relier le nouveau marquage au mouvement et à l'exécution comme marquage courant.

Dans le formalisme sNets, nous avons utilisé le mécanisme des actions sémantiques pour spécifier ces opérations. Nous avons défini des actions sur quelques-unes des entités de la partie dynamique du méta-modèle (Figure 150) :

- *execute* : définit la séquence d'opérations globale (voir Figure 151)
- *newMove* : crée un nouveau mouvement
- *selectATransition* : sélectionne une transition parmi les transitions éligibles
- *firableTransitions* : renvoie la liste des transitions d'un marquage susceptibles d'être mises à feu
- *fireTransition* : crée un nouveau marquage en mettant à feu une transition

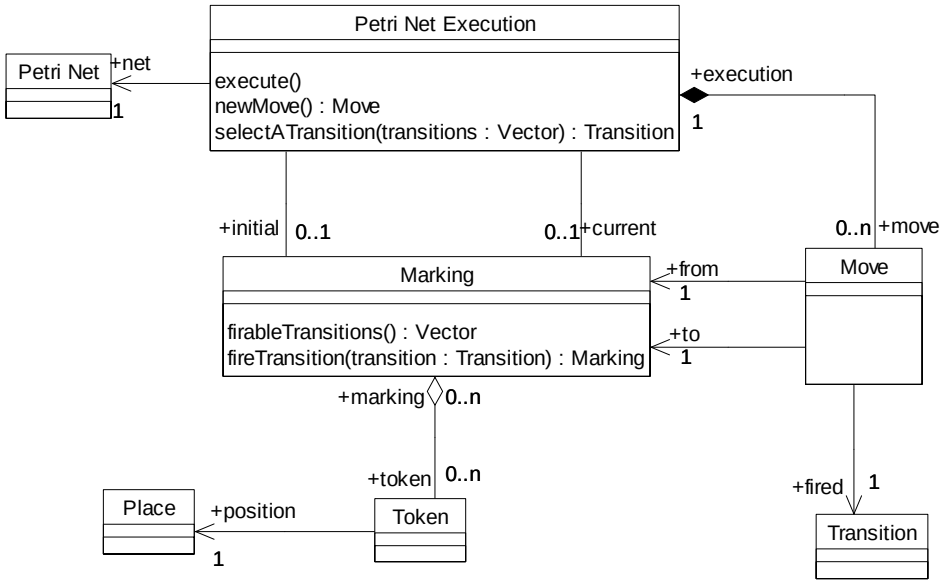


Figure 150. Les actions sémantiques définissant l'exécution d'un réseau de Petri

Ces actions ont été écrites en Java, car nous utilisions une implémentation Java de la sMachine. Le code de l'action *execute()* est le suivant :

```
// On récupère le marquage courant
SemNode currentMarking = current.getNodeForLink("current");

// On récupère les transitions éligibles
com.sun.java.util.collections.Vector firable =
        (com.sun.java.util.collections.Vector)
                ((sNets.actions.Actionable)currentMarking).action("firableTransitions");

// On élit la transition à mettre à feu
SemNode transitionToBeFired =
        (SemNode) ((sNets.actions.Actionable)current).actionWith("selectATransition",
                                                new Object[]{"transitions",firable});

// On crée un nouveau mouvement
SemNode newMove = (SemNode)((sNets.actions.Actionable)current).action("newMove");
current.setLinkToNode("move",newMove);
newMove.setLinkToNode("from", currentMarking);
newMove.setLinkToNode("fired", transitionToBeFired);

// On crée un nouveau marquage à partir du marquage courant
SemNode newMarking =
        (SemNode) ((sNets.actions.Actionable)currentMarking).actionWith("fireTransition",
                                        new Object[]{"transition",transitionToBeFired});
newMove.setLinkToNode("to",newMarking);
current.setLinkToNode("current",newMarking);
```

Figure 151. Code de l'action sémantique *execute*

Le méta-modèle des réseaux de Petri que nous avons défini devient ainsi exécutable. Si on invoque la méthode *execute()* sur une instance de *PetriNetExecution*, le graphe de marquage courant est remplacé par un nouveau graphe de marquage.

12.4. Action Semantics for MOF

Nous proposons donc, afin de pallier les manques des méta-modèles basés sur le MOF, d'intégrer dans le MOF un mécanisme similaire aux actions sémantiques sNets. Ces actions permettent de naviguer dans le modèle et de le mettre à jour, c'est-à-dire de créer ou de supprimer des entités et des liens et de modifier des attributs. On peut ainsi spécifier des traitements relativement complexes. La solution proposée dans les sNets présente toutefois un certain nombre de limites. Tout d'abord, elle est basée sur des langages de programmation. Elle est donc dépendante d'une plate-forme d'exécution. De plus ce langage est contextuel à l'implémentation de la sMachine. La portabilité des méta-modèles s'en trouve ainsi considérablement réduite. Pour pallier ces lacunes, il est donc nécessaire de définir un formalisme neutre d'un niveau d'abstraction plus élevé.

Les objectifs d'AS_PL sont assez similaires aux nôtres. Il s'agit de spécifier la sémantique des langages de programmation par le biais d'actions. Il pourrait donc être

intéressant de l'utiliser dans le cadre des langages de modélisation. Toutefois, nous considérons qu'il présente un certain nombre d'inconvénients. Tout d'abord, c'est un langage procédural, alors que les méta-modèles sont basés sur un paradigme objet. Ensuite, et surtout, c'est une notation basée sur une gammaire. L'utilisation d'AS_PL introduirait donc un nouveau formalisme au niveau du MOF, ce qui rendrait encore plus complexe le développement de méta-modèles. De plus, cela implique des mécanismes complexes pour contrôler la validité des actions avec le méta-modèle.

Une autre proposition que nous avons étudiée est AS_UML, bien qu'elle ne se place pas au même niveau que les actions sémantiques sNets. En effet, AS_UML fournit un formalisme pour la définition d'actions sur les modèles UML, et non sur les concepts d'UML. On peut prendre pour exemple les machines d'états UML. A tout état (*State*) d'un diagramme d'états UML, on peut relier une action à réaliser lors de l'entrée dans l'état (*doEntry*). AS_UML permet de spécifier le contenu de cette action. Par contre, le fait que cette action doive être lancée dès que l'état devient actif n'est pas formellement spécifié. Un des intérêts d'AS_UML tient au fait qu'il est défini dans le même formalisme qu'UML, ce qui facilite son intégration.

Une solution consiste donc à reprendre Action Semantics for UML et à le remonter au niveau du MOF. C'est ce qui a déjà été fait pour OCL, qui était à l'origine dédié à UML, et qui a été ensuite introduit dans le MOF. On disposerait alors de mécanismes permettant de spécifier des actions sur chacune des entités d'un méta-modèle, et ainsi de le rendre opérationnel. Etant donné que le MOF est très proche du noyau d'UML, cet enrichissement ne pose pas de problème majeur, ainsi que le montre la Figure 152. En effet, le MOF introduit déjà le concept d'*Operation*, qui définit un comportement pouvant être invoqué. Nous avons donc intégré au MOF les concepts de base d'AS_UML. Ces extensions sont représentées en grisé. Nous avons omis l'ensemble des entités qui spécialisent l'entité *Action* afin de ne pas surcharger le schéma. On voit donc que les ajouts au niveau du MOF sont relativement minimes. Ce que nous avons défini pourrait s'appeler « Action Semantics for MOF ».

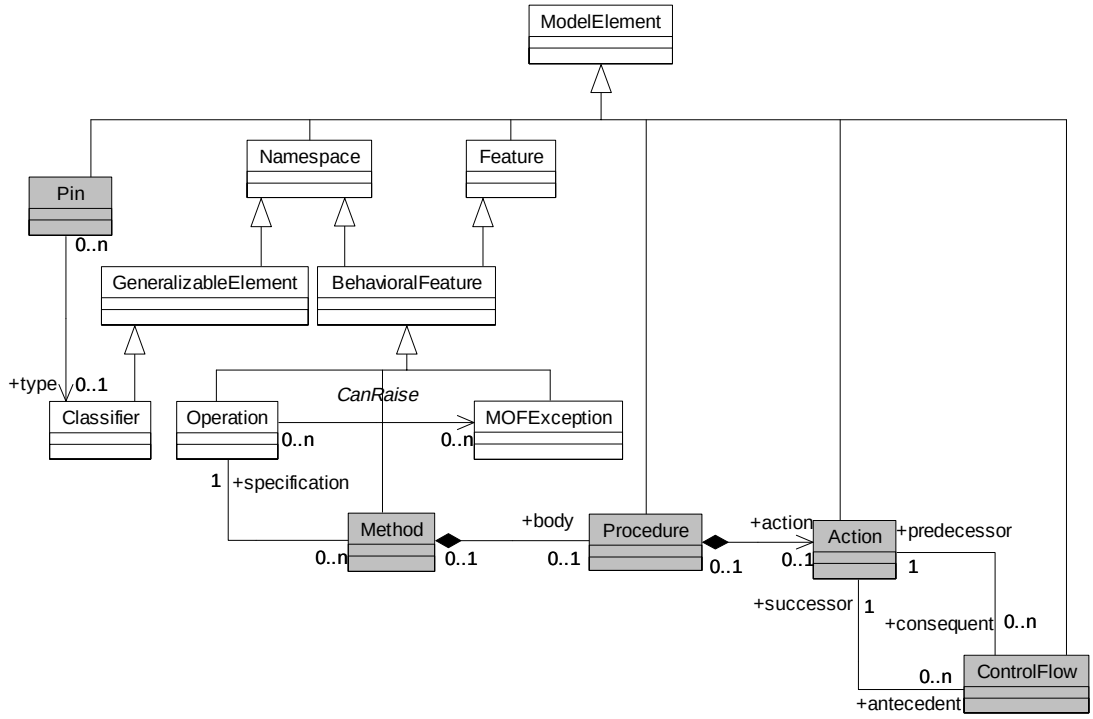


Figure 152. Intégration des actions dans le MOF (les extensions sont en grisé)

12.5. Exemple

Nous avons vu précédemment comment l'action *execute* pouvait être codée dans un environnement sNets Java. A l'aide d'un mécanisme tel que Action Semantics for MOF, nous pouvons la représenter dans un environnement MOF. La Figure 153 montre une partie de son implémentation. On commence par récupérer l'objet courant, c'est-à-dire l'instance d'*Execution* en cours. Cela se fait grâce à une instance de l'action *ReadSelfAction*. A partir de cet objet courant, on peut alors récupérer le marquage courant, en passant en paramètre la fin d'association *current* à une instance de l'action *ReadLinkEndObjectAction*. On peut ensuite invoquer l'opération *firableTransitions* sur ce marquage par l'intermédiaire d'une instance de l'action *CallOperationAction*. Le résultat renvoyé par cette action est alors utilisé par une autre instance de l'action *CallOperationAction*. Il va servir d'argument à l'opération *selectATransition* appliquée sur l'objet courant qui sélectionne la transition à mettre à feu. Toutes ces actions sont présentes dans la Figure 153. Nous avons rajouté des notes pour répercuter les commentaires et les noms des variables de l'implémentation sNets. Les actions suivantes, qui permettent de créer des nouveaux liens et des nouveaux objets peuvent également être modélisées.

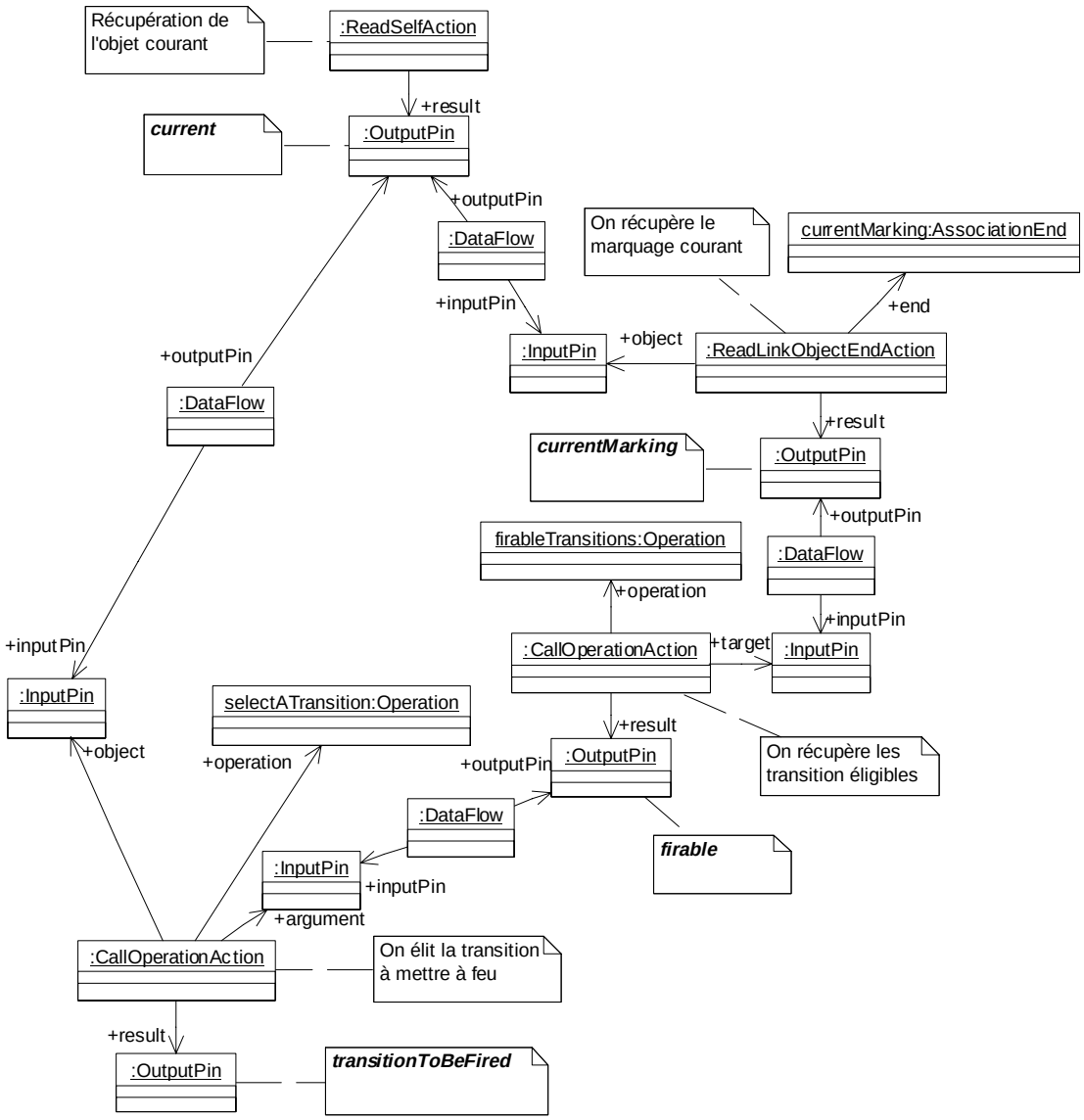


Figure 153. Modèle partiel de l'opération *execute*

Dans le chapitre précédent, nous avons identifié de façon informelle une partie statique et une partie dynamique dans les méta-modèles de processus. Les instances des concepts de la partie dynamique évoluent au cours du temps, ce qui n'est pas le cas de ceux de la partie statique. La spécification des règles d'évolution permet de marquer plus précisément cette distinction. En effet, on peut facilement définir les types des instances créés ainsi que ceux dont les instances voient leurs liens et leurs attributs être modifiés. Toutes ces actions étant explicitement représentées et intégrées avec les objets du modèle (classes, attributs, fin d'associations, opérations, etc.), on peut ainsi déterminer de façon automatique si tel concept appartient à la partie statique ou dynamique.

12.6. Conclusion

Il est communément considéré que les méta-modèles peuvent être découpés en trois grandes parties [2] :

- La partie terminologique : l'ensemble des concepts manipulés et les relations qui peuvent être établies entre eux
- La partie assertionnelle : des contraintes supplémentaires
- La partie pragmatique : un fourre-tout qui regroupe l'ensemble des aspects que l'on n'a pu formaliser et intégrer dans les deux premières parties

C'est dans cette dernière partie que se situe la sémantique. Notre proposition, Action Semantics for MOF, contribuerait donc à rationaliser un aspect de cette dernière partie. Doter le MOF d'un tel mécanisme ouvre de nombreuses perspectives. Tout d'abord, la façon dont un modèle s'exécute peut être précisément spécifiée. Les comportements des entités peuvent être précisément définis, communiqués et validés. Cela évite des interprétations divergentes. Les modèles basés sur n'importe quel méta-modèle peuvent alors être simulés, ce qui permet une validation rapide. De plus, les méta-modèles standards peuvent être étendus, non seulement au niveau des parties terminologiques et assertionnelles, mais également au niveau de la sémantique d'exécution.

Des travaux sont actuellement en cours pour la définition d'une machine virtuelle UML se basant sur les actions sémantiques ([79], [81]). Ces machines virtuelles permettront d'exécuter des modèles UML contenant des actions, permettant une vérification rapide des modèles de conception. Elever AS_UML au niveau du MOF permettrait de définir des machines virtuelles non seulement pour UML, mais pour n'importe quel méta-modèle. Par exemple, CWM définit tout un ensemble de concepts pour la transformation de modèle. Une transformation est réalisée par un ensemble d'étapes (*TransformationStep*) qui s'enchaînent. L'exécution ou la simulation d'une transformation CWM nécessite donc le développement d'une machine virtuelle dédiée. Une machine virtuelle basée sur le MOF pourrait éviter un tel développement, à condition que la cinématique d'une transformation ait été spécifiée dans le méta-modèle CWM, ainsi que le permettent les actions sémantiques.

Conclusion

Dans cette dernière partie, nous avons présenté un certain nombre de réflexions sur l'applicabilité des techniques de méta-modélisation à la définition de processus. Nous avons commencé par traiter de l'existence d'un ensemble de concepts communs aux formalismes de description de processus que nous avons étudiés. Après les avoir comparés, nous avons proposé notre vision de ce noyau basé sur les concepts suivants : processus, activité, structure de contrôle (transition et contrainte) et objet.

Le point suivant a concerné le couplage des méta-modèles de processus avec d'autres types de méta-modèles. Nous avons identifié deux relations : l'intégration et la correspondance structurelle. L'intégration consiste à établir des associations entre concepts définis dans des méta-modèles différents. En effet, le concept d'objet tel qu'il est spécifié dans les méta-modèles de processus est souvent des plus sommaire, ce qui peut présenter des limites à son utilisation. Afin de pallier ces lacunes sans compliquer le méta-modèle de processus, une solution consiste à mettre en relation ce dernier avec un autre méta-modèle plus adapté. Nous avons vu que cette mise en relation pouvait être faite à l'aide des mécanismes d'extension. La seconde relation reprend le constat fait par plusieurs auteurs d'une correspondance dans les formes du processus et du produit. Cela a donné lieu à quelques patterns exprimés de façon informelle. Nous avons vu que des mécanismes de transformation de modèles permettaient de les préciser.

Nous avons ensuite traité de la définition des aspects statiques et dynamiques des processus. A travers nos expérimentations, nous avons pu montrer comment ceux-ci pouvaient être pris en compte et intégrés. Nous avons ainsi pu établir des relations entre les concepts dédiés à la définition du processus et ceux qui représentent une exécution de ce processus. Nous avons également pu spécifier des contraintes sur les entités de la partie dynamique en fonction de celles de la partie statique. Nous proposons donc ici une autre vision que celle véhiculée par l'architecture à quatre niveaux traditionnelle, puisque l'exécution d'un processus devient une méta-entité explicite et n'est plus unifiée avec la relation d'instanciation.

Enfin, dans le dernier point, nous nous sommes intéressés à la spécification du comportement des processus. Nous avons vu que le MOF n'apporte actuellement pas de solution. Afin de pallier cette lacune, nous avons étudié l'utilisation d'un mécanisme basé sur les actions. Ce mécanisme a déjà été introduit dans le formalisme sNets, mais il présente quelques limites. Notre proposition consiste donc à intégrer Action Semantics for UML au niveau du MOF. Nous avons montré la faisabilité d'une telle extension du MOF.

13. Conclusion générale

13.1. Contributions

Notre travail de thèse a tendu à démontrer la pertinence des techniques de méta-modélisation dans le domaine des processus. Cela a pu être tout particulièrement mis en évidence par nos réalisations industrielles. Notre contribution scientifique a ensuite consisté à expliciter un certain nombre de problèmes et à proposer des solutions conceptuelles. Celles-ci n'ont pu être totalement abouties dans le cadre de cette thèse. Elles constituent à nos yeux des pistes de recherche intéressantes et prometteuses.

13.1.1. Représentation de processus

Dans cette thèse, nous avons montré l'applicabilité des techniques de méta-modélisation à la représentation de processus. Nous avons pu ainsi exprimer différents formalismes basés sur KIF, XML ou d'autres langages sous la forme de méta-modèles. Comparé à ces langages, l'utilisation d'une notation graphique facilite grandement leur lisibilité et leur compréhensibilité. En exprimant l'ensemble de ces formalismes sous la forme de méta-modèles basés sur le MOF, nous avons pu plus facilement faire émerger des structures similaires. C'est ce que nous nous sommes attachés à faire, en alignant ces différents formalismes et en proposant notre vision des concepts constituant le cœur des méta-modèles de processus.

13.1.2. Couplage produit processus

Des couplages forts ont souvent été soupçonnés entre les espaces de produit et les espaces de processus. A partir de différentes sources, ainsi que de notre expérience personnelle, nous avons pu mettre en évidence deux types de relations : l'intégration et la correspondance structurelle. Nous avons montré qu'elles peuvent être explicitées à l'aide des techniques de méta-modélisation. L'intégration peut être simplement définie par une mise en relation de méta-modèles. La correspondance structurelle nécessite des mécanismes plus complexes de transformation de modèles et de traçabilité. Les différentes aspects d'une entreprise peuvent être représentée par des méta-modèles différents mais interdépendants, c'est la dimension horizontale de l'architecture des modèles d'une entreprise proposée par D'Souza [22]. Nos travaux constituent donc un premier effort d'organisation de cette dimension horizontale.

13.1.3. Exécutabilité des modèles de processus

La plupart des formalismes de définition de processus ne fournissent pas de mécanismes permettant de représenter la manière dont ils s'exécutent. L'exécution de processus est parfois considérée comme un énième avatar de la relation d'instanciation. Elle peut également être définie séparément des concepts de définition. Nous avons donc proposé sa définition explicite à l'aide des techniques de méta-modélisation. Nous avons identifié deux parties distinctes au sein des méta-modèles de processus, la partie statique, qui regroupe les concepts permettant de spécifier les règles d'exécution (découpage et ordonnancement des activités, profil des acteurs, etc.) et la partie dynamique, qui introduit les concepts représentant une exécution particulière d'un processus. A travers nos expérimentations nous avons établi des relations entre ces deux parties. Nous avons également proposé la définition d'un formalisme d'action afin de spécifier précisément les évolutions des modèles d'exécution de processus.

13.1.4. Modélisation de processus logiciels

Afin de permettre la collecte et l'organisation des savoir-faire des experts métier de Sodifrance, nous avons proposé PMM, un méta-modèle dédié à la représentation des processus de développement logiciels que sont la maintenance et la migration. Nous avons démontré sa pertinence à travers la modélisation d'un processus complet de tierce maintenance applicative. Ce méta-modèle a également été appliqué pour représenter des processus bancaires ainsi que des processus de développement logiciels. Nous avons d'ailleurs présenté dans ce document une version du RUP basée sur PMM. Cette dernière application n'est pas surprenante au vu des nombreux points communs identifiables entre PMM et SPEM : hiérarchie assez similaire des activités, modularité par paquetages et processus, définition des concepts de rôles, d'informations et de documents, etc.

13.1.5. Ingénierie du processus

Dans un article sur le génie logiciel, Leon Osterweil [76] affirme que le processus de développement logiciel est également du logiciel (« Software process is software, too »). Dans une dernière version [77], il précise que la mise au point du processus de développement et le support de son exécution doivent bénéficier des techniques qui ont été développées pour la production de logiciel. Notre travail corrobore donc cette affirmation en ce qui concerne le processus de TMA, en l'adaptant au nouveau contexte défini par le MDA : séparation des aspects purement métier des spécificités dues à la plate-forme d'exécution, transformation de modèles et génération de code. Des prototypes ont également été réalisés pour des processus de migration et de développement logiciel, ainsi que sur des processus bancaires.

Nous avons appliqué ces nouvelles techniques afin de définir une ingénierie de processus dirigée par les modèles. Ces expérimentations ont été faites à travers nos différentes réalisations industrielles. Nous avons développé un certain nombre de services permettant d'exploiter les modèles basés sur PMM. A partir de modèles de processus nous avons pu générer de la documentation ou alimenter un outil de planification. Nous avons

également pu rendre ces modèles opérationnels par la génération de code exécutable par un moteur de workflow. Cette approche a été validée sur trois moteurs de workflow différents (FlowMind, W4 et Domino Workflow).

Lors des différents projets que nous avons pu mener, nous avons tenté de tirer parti de ces différents mécanismes pour faciliter la définition d'un processus. Cette définition se fait en collaboration avec les experts métier sur la base du modèle. L'utilisation d'un langage graphique permet en effet une plus grande implication des non-informaticiens dans le projet. Toutefois, il arrive fréquemment que la validation de tels modèles soit complexe. C'est pourquoi nous avons défini une démarche itérative, tant dans la définition du modèle que dans celle des services de génération, permettant à tout moment de valider un modèle même incomplet par l'emploi de services de prototypage. Ces services de génération peuvent être adaptés sur chacun des sites pour répondre aux spécificités techniques ou fonctionnelles.

13.2. Extensions et perspectives

13.2.1. Prise en compte des nouvelles spécifications de l'OMG

En parallèle de nos travaux, le monde industriel a beaucoup évolué. Beaucoup de ces évolutions sont à mettre au compte de l'OMG. Nous avons déjà évoqué le MDA, nouvelle approche du génie logiciel. Dans le domaine des processus, de nombreuses propositions ont également vu le jour ou sont en cours de finalisation. On peut citer SPEM et EDOC dont les publications de spécification finale ont eu lieu en février 2002. On peut également citer les travaux sur un méta-modèle de processus de workflow qui sont actuellement en cours d'élaboration à l'OMG, les propositions initiales ayant été reçues en janvier 2002. Ces spécifications sont appelées à devenir des standards reconnus et utilisés dans le domaine de l'industrie.

La solution que nous avons développée au cours de cette thèse est basée sur un méta-modèle propriétaire. Bien qu'il semble répondre aux besoins de modélisation des experts métier de Sodifrance, il n'aura ni la reconnaissance, ni la pérennité des spécifications de l'OMG. Il nous semble donc qu'une des premières suites de nos travaux consiste à prendre acte de l'émergence de ces standards et à étudier leur intégration et la pertinence de l'outillage que nous avons pu développer dans ce nouveau contexte.

Des premiers travaux exploratoires ont déjà été démarrés. En effet, Sodifrance est impliqué dans le projet RNTL TRAMs, Transformations Reposant sur une Architecture à base de Méta-Modèles pour la Migration des Systèmes d'Information vers le Web (http://www.industrie.gouv.fr/rntl/AAP2001/Fiches_Resume/TRAMS.htm). Ce projet est une collaboration entre EDF, France Télécom, l'IRIT, le LIP6 et Sodifrance. TRAMs définit un cadre méthodologique et une architecture logicielle (framework) pour maîtriser et optimiser le processus de migration d'un S.I. industriel ou commercial vers le Web. La solution TRAMs repose sur la méta-modélisation et la transformation de modèles, avec une démarche à base

de patterns de processus réutilisables et un framework générique, modulaire, ouvert, indépendant des outils. La définition de la méthodologie passe par l'analyse et la classification des processus actuels mis en œuvre dans les entreprises partenaires. Dans ce cadre, le choix du formalisme pivot permettant d'échanger des modèles de processus et de stocker des patterns de processus s'est tout naturellement porté sur SPEM. En tant que partenaire industriel et outilleur du projet, Sodifrance sera amené à réutiliser l'ensemble des concepts et des outils développés dans cette thèse.

13.2.2. Intégration de l'ingénierie de processus dans un cadre plus étendu

Nos travaux se sont surtout concentrés sur la définition et l'automatisation des processus. Toutefois, la dimension processus n'est pas l'unique aspect d'une entreprise. On y trouve également la dimension produit, la structure organisationnelle. Nous avons ainsi pu constater de nombreux couplages entre ces différents aspects. Nous avons également proposé des mécanismes permettant de les formaliser (mise en relation, transformation). Ainsi, le développement du système d'information doit être une approche globale. La définition des processus en est un point important, mais d'autres points méritent également d'être pris en compte. La démarche suivie doit alors tenir compte des interactions existant entre les différents aspects. Des méthodes telles que SSADM (Structured Systems Analysis and Design Methodology) peuvent être à ce titre des sources d'inspiration. Intégrer nos travaux dans un tel cadre nous semble constituer une extension logique.

13.2.3. Implémentation d' « Action Semantics for MOF »

Dans notre travail sur la définition de la sémantique d'exécution de la sémantique d'exécution des modèles de processus, nous avons proposé d'élever Action Semantics for UML au niveau du MOF. Ce dernier serait ainsi pourvu d'un mécanisme similaire aux actions sémantiques sNets. Le comportement des entités dynamiques d'un méta-modèle pourrait ainsi être précisément spécifié, permettant :

- L'échange des points de vue entre les concepteurs du méta-modèle,
- La spécification des modifications apportées par des extensions,
- Une meilleure information pour les utilisateurs du méta-modèle,
- La mise au point d'outils génériques pour la simulation de modèles, quel que soit le méta-modèle.

Une première validation de cette approche consiste donc à implémenter les spécifications d'Action Semantics for UML dans un environnement MOF. Ce travail n'a pu être mené au cours de la thèse. Nous pensons qu'il peut constituer une expérience intéressante. L'expression de la sémantique d'exécution par les actions n'est bien sûr pas l'unique solution. En particulier, nous avons régulièrement comparé les concepts d'exécution et de transformation, l'exécution d'un modèle étant une suite de transformations au sein du même formalisme. Il serait donc également intéressant d'étudier si des mécanismes de transformation peuvent fournir les mêmes fonctionnalités qu'un langage d'actions.

Table des figures

Liste des publications

Bézivin J., Bouchet J.P., Breton E., « Revisiting the P&P Pattern with Explicit Meta-Models », INCOSE/LA Spring Conference, Pasadena, USA, avril 1999.

Breton E., Bézivin J., « An Overview of Industrial Process Meta-Models », *Proceedings of the 13th International Conference Software & System Engineering and their Applications (ICSSEA'2000)*, Paris, France, décembre 2000, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/icssea2.pdf.

Breton E., Bézivin J., « Using Meta-Model Technologies to Organize Functionalities for Active System Schemes », *Proceedings of the Workshop of Ontologies in Agent Systems (OAS 2001)*, 5th International Conference on Autonomous Agents, Montreal, Canada, mai 29, 2001, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-52/oas01-breton.pdf.

Breton E., Bézivin J., « Process-Centered Model Engineering », *Proceedings of the5th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2001)*, Seattle, Washington, USA, septembre 2001, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/edoc.pdf.

Breton E., Bézivin J., « Model Driven Process Engineering », *Proceedings of the 25th Annual International Computer Software and Application Conference (COMPSAC 2001)*, Chicago, Illinois, octobre 2001, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/compsac.pdf.

Breton E., Bézivin J., « Towards an Understanding of Model Executability », *Proceedings of the Second International Conference on Formal Ontology in Information Systems (FOIS'2001)*, Ogunquit, Maine, USA, octobre 2001, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/fois.pdf.

Breton E., Bézivin J., « Un méta-modèle de gestion par les activités », *Actes du colloque Coopération Innovation et Technologies (CITE 2001)*, Troyes, France, novembre 2001, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/cite.pdf.

Breton E., Bézivin J., « Weaving Definition and Execution Aspects of Process Meta-Models », *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35) - Minitrack Software Technology/Integrated Modeling of Distributed Systems and Workflow Applications*, Waikoloa, Hawaii, janvier 2002, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/hicss.pdf.

Breton E., Ouertal M., Bouchet J.P., « Executing process models using a workflow engine », *Proceedings of the 13th International Conference Software & System Engineering and their Applications (ICSSEA'2000)*, Paris, décembre 2000, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/icssea1.pdf.

Bibliographie

[1] Akazi, « Sodifrance renforce sa qualité de service avec FlowMind », 2001, http://www.akazi.fr/societe/ref/sp/sodifrance.pdf.

[2] Bézivin J., « Who's Afraid of Ontologies? », *Proceedings of OOPSLA'98 Workshop: Model Engineering, Methods and Tools Integration with CDIF*, Vancouver, October 1998, http://www.metamodel.com/oopsla98-cdif-workshop/bezivin1/.

[3] Bézivin J., « From Object Composition to Model Transformation with the MDA », http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda/TOOLS.USA.pdf.

[4] Bézivin J., Gerbé O., « New Trends in Applied Model Engineering », submitted for publication, 2001, http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda/OFTA/new.trends.pdf.

[5] Bézivin J., Lemesle R., « The sBrowser: a prototype Meta-Browser for Model Engineering », OOPSLA'98, Workshop on Model Engineering, Methods and Tools Integration with CDIF, Vancouver, octobre 1998, http://www.metamodel.com/oopsla98-cdif-workshop/bezivin2/.

[6] Bézivin J., Lemesle R., « Reflective Modeling Schemes », *Proceedings of OOPSLA'99 workshop on Object-Oriented Reflection and Software Engineering, pp 107-122*, Denver, USA, novembre 1999, ftp://ftp.disi.unige.it/pub/person/CazzolaW/OORaSE99/107-122%20Bezivin.ps.gz.

[7] Bézivin J., Lemesle R., « Towards A True Reflective Scheme Reflection and Software Engineering », *LNCS Vol. 1826 (pp 21-38)*, Springer, 2000.

[8] Bézivin J., Ploquin N., « Combining the Power of Meta-Programming and Meta-Modeling within the OMG MDA framework », présentation faite au 2nd workshop on UML for Enterprise Applications: Model Driven Solutions for the Enterprise, San Francisco, décembre 2001.

[9] BPMI.org, « Business Process Modeling Language (BPML) », mars 2001, http://www.bpmi.org.

[10] Breton E., Bézivin J., « Model Driven Process Engineering », *Proceedings of the 25th Annual International Computer Software and Application Conference (COMPSAC 2001)*, Chicago, Illinois, octobre 2001, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/compsac.pdf.

[11] Breton E., Bézivin J., « Un méta-modèle de gestion par les activités », *Actes du colloque Coopération Innovation et Technologies (CITE 2001)*, Troyes, France, novembre 2001, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/cite.pdf.

[12] Breton E., Ouertal M., Bouchet J.P., « Executing process models using a workflow engine », *Proceedings of the 13th International Conference Software & System Engineering and their Applications (ICSSEA'2000)*, Paris, décembre 2000, http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/EB/Publications/icssea1.pdf.

[13] Chandrasekaran B., Josephson J. R., Richard Benjamins V., « The ontology of Tasks and Methods », *Proceedings of the 11th Knowledge Acquisition Modeling and Management Workshop (KAW'98)*, Banff, Canada, avril 1998, http://www.cis.ohio-state.edu/~chandra/Ontology-of-Tasks-Methods.PDF.

[14] Conradi R., Fernström C., Fuggetta A., Snowdon R., « Towards a Reference Framework for Process Concepts », in *Proceedings of the 2nd European Workshop on Software Process Technology (EWSPT'92)*, Trondheim, Norvège, septembre 1992, http://www.idi.ntnu.no/~epos/Papers/pm-concepts-ewspt92.ps.

[15] Coplien J., « A Development Process Generative Pattern Language », *Proceedings of PloP 94*, Monticello, Italie, août 1994, ftp://st.cs.uiuc.edu/pub/patterns/plop-papers/ProcessPatterns.ps.

[16] Cook S., « Model-driven approaches to software development », présentation au colloque OCM 2002, Nantes, France, mars 2002.

[17] Crowston K., « A Taxonomy of Organisational Dependencies and Coordination Mechanisms », Technical Report 174, Massachusetts Institute of Technology, août 1994, http://ccs.mit.edu/papers/CCSWP174.html.

[18] Curtis B., Kellner M., Over J., « Process Modelling », *Communications of the ACM 35*, 1992.

[19] Data Access Technologies, DSTC, IBM, Open-IT, Unisys, « Human-Usable Textual Notation », OMG Document ad/2002-03-02, avril 2002, http://cgi.omg.org/cgi-bin/doc?ad/02-03-02.

[20] Derniame J.-C., Kaba B. A., Wastell D., *Software Process: Principles, Methodology, and Technology*, Springer, 1999.

[21] Dominguez E., Rubio A., Zapata M., « Meta-modelling of Dynamic Aspects: The Noesis Approach », *Proceedings of ECOOP'2000 Workshop: International Workshop on Model Engineering*, Nice, juin 2000, http://www.metamodel.com/IWME00/articles/rubio.pdf.

[22] DSouza D., « Model-Driven Architecture and Integration – Opportunities and Challenges », mars 2001, ftp://ftp.omg.org/pub/docs/ab/01-03-02.pdf.

[23] Dumas M., ter Hofstede A., « ULM Activity Diagrams as a Workflow Specification Langage», *Proceedings of the International Conference on the Unified Modeling Language (UML)*, Toronto, Canada, octobre 2001, http://sky.fit.qut.edu.au/~dumas/uml01_dumas.pdf.

[24] ebXML Business Process Project Team, « ebXML Business Process Specification Schema Version 1.01 », mai 2001, http://www.ebxml.org/specs/ebBPSS.pdf.

[25] Electronic Industries Association, « CDIF CASE Data Interchange Format - Overview », EIA/IS-106, 1994, http://www.eigroup.org/cdif/how-to-obtain-standards.html.

[26] Fadel F., Fox M., Gruninger M., « A Generic Enterprise Resource Ontology », *Proceedings of the Third Workshop on Enabling Technologies - Infrastructures for Collaborative Enterprises*, West Virginia University, 1994, http://www.eil.utoronto.ca/EIL/public/res_ontology.ps.

[27] Fensel D., van Harmelen F., « A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise », *The Knowledge Engineering Review*, 1994.

[28] Fox M., Barbuceanu M., Gruninger M., « An Organisation Ontology for Enterprise Modelling: Preliminary Concepts for Linking Structure and Behaviour », *Computers in Industry*, Vol. 29, pp. 123-134, 1996, http://www.eil.utoronto.ca/EIL/public/comind.ps.

[29] Georgakopoulos D., Hornick M., Sheth A., « An overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure », *Distributed and Parallel Databases 3 (2)* , 1995, http://citeseer.nj.nec.com/cache/papers2/cs/2020/http:zSzzSzra.cs.uga.eduzSz~amitzSz63-dapd-Dimitris.pdf/georgakopoulos95overview.pdf.

[30] Giese H., « Towards a Dynamic Model for the UML », *Proceedings of 14th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications, Workshop: Rigorous Modeling and Analysis with the UML: Challenges and Limitations*, novembre 1999, http://www.upb.de/cs/hg/archive/1999/OOPSLA99/oopsla99.html.

[31] Gruber T.R., « Towards Principles for the Design of Ontologies Used for Knowledge Sharing », *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, 1993, http://citeseer.nj.nec.com/cache/papers/cs/490/http:zSzzSzwww-ksl.stanford.eduzSzknowledge-sharingzSzpaperszSzonto-design.pdf/gruber93toward.pdf.

[32] Gruninger M., Fox M., « An Activity Ontology for Enterprise Modelling », Third IEEE Workshop on Enabling Technologies - Infrastructures for Collaborative Enterprises, University of West Virginia, 1994, http://www.ie.utoronto.ca/EIL/public/actontAAAI94.ps.

[33] Harel D., Rumpe B., « Modeling Languages: Syntax, Semantics and All That Stuff – Part 1: The Bsic Stuff », Research Report MCS00-16, août 2000, http://www.wisdom.weizmann.ac.il:81/Dienst/Repository/2.0/Body/ncstrl.weizmann_il/MCS00-16/postscript.

[34] Hoare C.A.R., *Communicating Sequential Processes*, Prentice-Hall International, 1985.

[35] Hollingsworth D., « Workflow Management Coalition The Workflow Reference Model », Document WfMC-TC-1003, novembre 1994, http://www.wfmc.org/standards/docs/tc003v11.pdf.

[36] Hruby P., « Specification of Workflow Management Systems with UML », *Proceedings of the 1998 OOPSLA Workshop on Implementation and Application of Object-oriented Workflow Management Systems*, Vancouver, Canada, 1998, http://citeseer.nj.nec.com/cache/papers/cs/5575/http:zSzzSzwwwdb.inf.tu-dresden.dezSzOOPSLA98-Workflow-WorkshopzSzhrub98a.pdf/hruby98specification.pdf.

[37] ISO, « Guide sur l'approche processus appliquée aux systèmes de management de la qualité », Document ISO/TC 176/SC 2/N544R, mai 2001, http://www.iso.ch/iso/fr/iso9000-14000/iso9000/2000rev9.html.

[38] ISO, « Industrial automation system and integration – Process Specification Language », Document ISO/CD 18629-11, février 2002, http://www.tc184-sc4.org/SC4_Open/Projects/Files/N262%20CD%2018629-11final.pdf.

[39] ISO, « Information Technology – Information Resource Dictionary System (IRDS) framework », Document ISO/IEC 10027:1990, 1990, http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17985.

[40] ISO, « Life Cycle Management – System Life Cycle Processes », Document ISO/IEC 15288, janvier 2000, http://www.incose.org/stc/news_ISO15288cd2.htm.

[41] ISO, « XML representation of EXPRESS schemas and data », Document ISO/TC184/SC 4/WG11 N140, octobre 2000, http://www.nist.gov/sc4/step/parts/part028/pdts/p28_n140.zip.

[42] Jackson M., *System Development*, Prentice-Hall International, 1983.

[43] Jacobson I., Ericsson E., Jacobson A., *The ObjectAdvantage – Business Process Reengineering with Object Technology*, Addison-Wesley, 1994.

[44] Kim H., Fox M., « Formal Models of Quality and ISO 9000 Compliance: An Information Systems Approach », *American Quality Congress (AQC) Conference, American Society for Quality Control*, Las Vegas NV, 1994, http://www.eil.utoronto.ca/EIL/public/ASQCPaper12-93.ps.

[45] Kruchten P., *The Rational Unified Process – an introduction*, Addison-Wesley, 1999.

[46] Lee J., Grunninger M., Jin Y., Malone T., Tate A., Yost G. & other members of the PIF Working Group, « The PIF Process Interchange Format and Framework Version 1.2 », *The Knowledge Engineering Review, Vol. 13, No. 1, pp. 91-120*, Cambridge University Press, mars 1998, http://spot.colorado.edu/~jintae/pif/pif12.rtf.

[47] Lei Y., Singh M.P., « A Comparison of Workflow Metamodels », *Proceedings of the ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling*, UCLA, Los Angeles, California, novembre 1997, http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/metamodels.pdf.

[48] Lemesle R., « Transformation Rules Based on Meta-Modeling », *Proceedings of EDOC'98*, San Diego, novembre 1998, http://iae.univ-nantes.fr/recherch/travaux/cahiers98/lemesle.html.

[49] Lemesle R. « Techniques de modélisation et de Méta-Modélisation », thèse de l'Université de Nantes, octobre 2000, http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda/richard.pdf.

[50] Leymann F., « Web Service Flow Language (WSFL 1.0) », mai 2001, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[51] Lissandre M., *Maîtriser SADT*, Armand Colin, 1990.

[52] Lorino P., *Le contrôle de gestion stratégique*, Dunod, 1991.

[53] Loos P., Allweyer T., « Process Orientation and Object-Orientation – An Approach for Integrating UML and Event-Driven Process Chains », *Publication of the Institut für Wirtschaftsinformatik*, Saarbrücken, Germany, mars 1998, http://www.tu-chemnitz.de/wirtschaft/wi2/home/loos/iwih144.pdf.

[54] Loos P., Allweyer T., « Process Orientation in UML through Integration of Event-Driven Process Chains », *Proceeding of UML'98: Beyond the Notation First International Workshop*, Mulhouse, France, juin 1998.

[55] Malone T., Crowston K., « The Interdisciplinary Study of Coordination », Technical Report 157, Massachusetts Institute of Technology, novembre 1993, http://ccs.mit.edu/papers/CCSWP157.html.

[56] Mayer R., Menzel C., Painter M., de Witte P., Blinn T., Perakath B., « Information Integration for Concurrent Enginnering (IICE) – IDEF3 Process Description Capture Method Report », septembre 1995, http://www.idef.com/Downloads/pdf/Idef3_fn.pdf.

[57] Mellor S., Tockey S., Artaud R., Leblanc P., « Software-platform-independent, Precise Action Specifications for UML », *Proceeding of UML'98: Beyond the Notation First International Workshop*, Mulhouse, France, juin 1998, http://www.kc.com/as_site/download/UML_AS_paper.pdf.

[58] Mevellec P., *Outils de Gestion – La pertinence retrouvée*, Editions Comptables Malesherbes, 1990.

[59] Meyer B., *Introduction to the Theory of Programming Languages*, Prentice Hall International, 1990.

[60] Milner R., *Communication and Concurrency*, Prentice-Hall International, 1989.

[61] Morel G., Kassel G., « Définition d'un cadre conceptuel pour rapprocher L'Ingénierie des Connaissances, la Représentation des Connaissances Objet et le Génie Logiciel Objet », *Actes de la Journée Jeunes Chercheurs Ingénierie des Systèmes d'Information et Ingénierie des Connaissances*, Paris, décembre 1999.

[62] Mosses P., *Action Semantics,* Cambridge University Press, 1992.

[63] OMG, « Meta Object Facility (MOF) Specification », OMG Document formal/2000-04-03, mars 2000, http://www.omg.org/technology/documents/formal/mof.htm.

[64] OMG, « Organizational Structure Facility Specification », OMG Document dtc/2001-09-01, août 2001, http://cgi.omg.org/cgi-bin/doc?dtc/01-09-01.

[65] OMG, « The Software Process Engineering Metamodel (SPEM) », OMG Document ad/2001-06-05, juin 2001, http://cgi.omg.org/cgi-bin/doc?ad/01-06-05.

[66] OMG, « UML 2.0 Diagram Interchange RFP. », OMG Document ad/2001-02-39, mars 2001, http://cgi.omg.org/cgi-bin/doc?ad/01-02-39.

[67] OMG, « Unified Modeling Language (UML), version 1.4 », OMG Document formal/2001-09-67, septembre 2001, http://www.omg.org/cgi-bin/doc?formal/01-09-67.

[68] OMG, « Unified Modeling Language Specification (ActionSemantics) », OMG Document ptc/02-01-09, janvier 2002, http://www.omg.org/cgi-bin/doc?ptc/2002-01-09.

[69] OMG, « UML Extensions for Workflow Process Definition Request for Proposal », OMG Document bom/2000-12-11, décembre 2000, http://cgi.omg.org/cgi-bin/doc?bom/00-12-11.

[70] OMG, « UML Profile and Interchange Models for Enterprise Application Integration (EAI) Specification », OMG Document ptc/02-02-02, février 2002, http://cgi.omg.org/cgi-bin/doc?ptc/2002-02-02.

[71] OMG, « UML Profile for CORBA Specification », OMG Document ptc/01-01-06, septembre 2000, http://cgi.omg.org/cgi-bin/doc?ptc/2001-01-06.

[72] OMG, « UML Profile for Enterprise Distributed Object Computing », OMG Document ptc/02-02-05, février 2002, http://cgi.omg.org/cgi-bin/doc?ptc/2002-02-05.

[73] OMG, « Workflow Resource Assignment Interfaces (RAI) RFP », OMG Document bom/2000-01-03, janvier 2000, ftp://ftp.omg.org/pub/docs/bom/00-01-03.pdf.

[74] OMG, « XMI Production for XML Schema Specification », OMG Document ptc/01-12-03, novembre 2001, http://cgi.omg.org/cgi-bin/doc?ptc/01-12-03.

[75] OMG, « XML Metadata Interchange (XMI) Specification v1.1 », OMG Document formal/2000-11-02, novembre 2000, http://www.omg.org/cgi-bin/doc?formal/2000-11-02.

[76] Osterweil, L., « Software Process is Software, Too », *Proceedings of the 9th International Conference on Software Engineering*, Monterey, Californie, 1987.

[77] Osterweil, L., « Software Processes Are Software Too, Revisited », *Proceedings of the 19th International Conference on Software Engineering*, Boston, Massachussets, 1997, ftp://ftp.cs.umass.edu/pub/techrept/techreport/1997/UM-CS-1997-020.ps.

[78] Paulk M., Curtis B., Chrissis M. B., Weber C., « Capability Maturity Model for Software, Version 1.1 », Technical Report CMU/SEI-93-TR-24, février 1993, http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.024.html.

[79] Pennaneach F., Sunye G., « Towards an execution engine for the UML », *Proceedings of UML'2000*, York, octobre 2000, http://www.disi.unige.it/person/ReggioG/UMLWORKSHOP/Pennaneach.pdf.

[80] Pease A., « Core Plan Representation, version 4 », novembre 1998, http://reliant.teknowledge.com/CPR2/Reports/CPR-RFC4/.

[81] Riehle D., Fraleigh S., Bucka-Lassen D., Omorogbe N., « The Architecture of a UML Virtual Machine », *Proceedings of the 2001 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '01)*, Tampa Bay, Floride, USA, octobre 2001, http://www.riehle.org/computer-science-research/2001/oopsla-2001.pdf.

[82] Rumbaugh J., *OMT – Modélisation et conception orientées objet*, Prentice Hall, 1994.

[83] Scacchi W., « Modeling, Integrating, and Enacting Complex Organizational Processes », K. Carley, L. Gasser, and M. Prietula (eds.), *Simulating Organizations: Computational Models of Institutions and Groups*, 153-168, MIT Press, 1998, http://cwis.usc.edu/dept/ATRIUM/Papers/Business_Process_Modeling.ps.

[84] Scheer A.-W., *ARIS – Business Process Frameworks*, Springer, 1998.

[85] Schlenoff C., Knutilla A., Ray S., « A Robust Process Ontology for Manufacturing Systems Integration », *Proceedings of 2nd International Conference on Engineering Design and Automation*, Maui, Hawaii, août 1998, http://www.ontology.org/main/papers/psl.html.

[86] Schulze W., « Fitting the Worflow Management Facility into the Object Management Architecture », *Proceedings of the Third OOPSLA Workshop on Business Object Design and Implementation*, Atlanta, USA, octobre 1997, http://wwwdb.inf.tu-dresden.de/dokumente/ls-dokumente/schu97d.pdf.

[87] Singh M.P., « Semantic Considerations on Workflows: An Algebra for Intertask Dependencies », *Proceedings of 5th International Workshop on Database Programming Languages (DBPL)*, Gubbio, Umbria, Italy, septembre 1995, http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/dbpl-95.pdf.

[88] Singh M.P., « Formal Aspects of Workflow Management - Part 1: Semantics », *Technical Report TR-96-08*, Department of Computer Science, North Carolina State University, January 1996, http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/wf_semantics.ps.gz.

[89] Sodifrance, « Scriptor 2.3 - Manuel de l'utilisateur », document technique, 2001.

[90] Sodifrance, « Scriptor-Transformation - Manuel de l'utilisateur », document technique, 2001.

[91] Sodifrance, « Workflow Builder 2.4 - Manuel de l'utilisateur », document technique, 2001.

[92] Soley R. and the OMG Staff Strategy Group, « Model Driven Architecture », novembre 2000, ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf.

[93] Sun Microsystems, « Java Metadata Interface (JMI) Specification – JSR 040 », novembre 2001, http://jcp.org/aboutJava/communityprocess/review/jsr040/.

[94] Sunye G., Pennaneac'h F., Ho W.-M., Le Guennec A., Jézéquel J.-M., « Using UML Action Semantics for Executable Modeling and Beyond », *Proceedings of Advanced Information Systems Engineering --- CAiSE 2001*, Interlaken, Suisse, juin 2001, http://www.irisa.fr/triskell/publis/2001/Sunye01a.pdf.

[95] Tham K., Fox M., Gruninger M., « A Cost Ontology for Enterprise Modelling », *Proceedings of the Third Workshop on Enabling Technologies - Infrastructures for Collaborative Enterprises*,West Virginia University, 1994, http://www.eil.utoronto.ca/papers/abstracts/30.html.

[96] Thatte S., « XLANG – Web Services for Business Process Design », http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[97] Uschold M., « Where is the Semantics on the Semantic Web », Ontologies and Agents Workshop, Montreal, 29 mai 2001, http://cis.otago.ac.nz/OASWorkshop/.

[98] Uschold M., Gruninger M., « Ontologies: Principles, Methods and Applications », *Knowledge Engineering Review 11(2)*, 1996, http://citeseer.nj.nec.com/cache/papers/cs/3214/http:zSzzSzwww.cm.cf.ac.ukzSzUserzSzJ-C.PazzagliazSzReferenceszSzart:Uschold-96.pdf/uschold96ontologie.pdf.

[99] Van der Aalst W.M.P., Barros A.P., ter Hofstede A.H.M., Kiepuszewski B., « Advanced Workflow Patterns », 7th International Conference on Cooperative Information Systems (CoopIS 2000), volume 1901 of *Lecture Notes in Computer Science*, pp 18-29. Springer-Verlag, Berlin, 2000, http://tmitwww.tm.tue.nl/staff/wvdaalst/Publications/p105.pdf.

[100] W3C, « Extensible Stylesheet Language Version 1.0 », recommandation W3C, octobre 2001, http://www.w3.org/TR/xsl/.

[101] W3C, « Web Service Description Language (WSDL) 1.1 », note W3C, mars 2001, http://www.w3.org/TR/wsdl.

[102] W3C, « XML Path Language (XPath) Version 1.0 », recommandation W3C, 16 novembre 1999, http://www.w3.org/TR/xpath.

[103] W3C, « XSL Transformations (XSLT) Version 1.0 », recommandation W3C, 16 novembre 1999, http://www.w3c.org/TR/xslt.

[104] Warmer J., Kleppe A., *The Object Constraint Language Precise Modeling with UML,* Addison Wesley, octobre 1998.

[105] Wirth N., « On the Composition of Well-Structured Programs », International Computing Symposium 1973, Davos, Switzerland, septembre 1973.

[106] Workflow Management Coalition, « Interface 1: Process Definition Interchange Process Model », octobre 1999, http://www.wfmc.org/standards/docs/TC-1016-P_v11_IF1_Process_definition_Interchange.pdf.

[107] Workflow Management Coalition, « Terminology and Glossary », Document WfMC-TC-1011, février 1999, http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf.

[108] zur Mühlen M., « Evaluation of Workflow Management Systems Using Meta Models », *Proceedings of the 32nd Hawaii International Conference on System Sciences (HICSS-32)*, 1999.

Lexique

ABC	Activity-Based Costing
ABM	Activity-Based Management
ARIS	Architecture of Integrated Information Systems
BPMI	Business Process Management Initiative
BPML	Business Process Management Language
BPMS	Business Process Management System
CAM-I	Consortium for Advanced Manufacturing International
CCS	Calculus of Communicating Systems
CDIF	CASE Data Interchange Format
CIFRE	Conventions industrielles de formation par la recherche
CMM	Capability and Maturity Model
COM	Component Object Model (Microsoft)
CORBA	Common Object Request Broker Architecture
CPR	Core Plan Representation
CRGNA	Centre de Recherche en Gestion de Nantes Atlantique
CSP	Communicating Sequential Processes
CWM	Common Warehouse Metamodel
DARPA	Defense Advanced Research Project Agency
DTD	Document Type Definition
EAI	Enterprise Application Integration
EBNF	Extended Backus-Naur Form

ebXML	Electronic Business using eXtensible Markup Language
EDOC	Enterprise Distributed Object Computing
EJB	Enterprise Java Beans
EPC	Event-driven Process Chains
FlowMind	Moteur de workflow édité par la société Akazi
IDL	Interface Definition Language (OMG)
IRDS	Information Resource Dictionary System
ISO	International Organization for Standardization
JCL	Job Control Language
JSD	Jackson System Design
JSP	Jackson Structured Programming
JSR	Java Specification Request
KIF	Knowledge Interchange Format
LDAP	Lightweight Directory Access Protocol
MDA	Model Driven Architecture
MOF	Meta-Object Facility
OCL	Object Constraint Language
OMA	Object Management Architecture
OMG	Object Management Group
OMT	Object Modeling Technique
PDCA	Plan-Do-Check-Act
PERT	Program Evaluation and Review Technique
PIF	Process Interchange Format
PIM	Platform-Independent Model
PMM	Process Meta-Model (méta-modèle de processus que nous avons conçu pour Sodifrance)

PSL	Process Specification Language
PSM	Platform Specific Model
RAI	Resource Assignment Interface
RFP	Request For Proposal
RUP	Rational Unified Process
RMI	Remote Method Invocation
RNTL	Réseau National des Technologies Logicielles
SADT	Structured Analysis and Design Technique
Scriptor-G	Scriptor-Generation, outil de génération de code développé par Sodifrance
Scriptor-T	Scriptor-Transformation, outil de transformation de modèles développé par Sodifrance
sNets	Système de méta-modélisation développé par l'Université de Nantes et finalisé lors d'un partenariat avec Sodifrance
SPEM	Software Process Engineering Metamodel
SSADM	Structured Systems Analysis and Design Methodology
TMA	Tierce Maintenance Applicative
TOVE	Toronto Virtual Enterprise
TRAMs	Transformations Reposant sur une Architecture à base de Méta-Modèles pour la Migration des Systèmes d'Information vers le Web
UML	Unified Modeling Language
W3C	World Wide Web consortium
WfMC	Workflow Management Coalition
WPDL	Worflow Process Definition Language
WSDL	Web Service Definition Language
WSFL	Web Service Flow Language
XMI	XML Model Interchange (OMG)
XML	eXtensible Markup Language

XML-sNets	Dialecte XML utilisé comme format d'échange dans l'environnement sNets
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation