Numéro d'ordre : **Université des Sciences et Technologies de Lille** **Année 2003**

THESE

présentée et soutenue publiquement le 16 décembre 2003

En vue de l'obtention du grade de

DOCTEUR DE
L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

Discipline : Informatique

par

Thomas VANTROYS

Du langage métier au langage technique, une plate-forme flexible d'exécution de scénarios pédagogiques

Composition du jury

Président :	Pr Laurence Duchien	LIFL, Université de Lille I
Rapporteurs :	Pr Michel Riveill	ESSI, Université de Nice
	Pr Pierre Tchounikine	LIUM, Université du Maine
Examinateurs :	Pr Pierre Dillenbourg	EPFL, Lausanne (Suisse)
	M Olivier Walbecq	Archimed, Lille
Directeur de thèse :	Pr Alain Derycke	CUEEP, Université de Lille I
Co-directeur de thèse :	Dr Yvan Peter	CUEEP, Université de Lille I

Laboratoire TRIGONE - Équipe NOCE

Remerciements

Mad, adj :
Affected with a high degree of intellectual independence ; [. . .]

Ambrose Bierce, ***The Devil's Dictionnary***.

Je tiens à remercier ici tous ceux qui ont ouvert la voie à ces trois années de folie douce et à tous ceux qui ont contribué de près ou de loin à ce cheminement.

Je tiens tout d'abord à remercier madame **Laurence Duchien**, Professeur des Universités au Laboratoire d'Informatique Fondamentale de Lille (LIFL), pour m'avoir fait l'honneur de présider le jury de cette thèse.

Je tiens également à remercier monsieur **Michel Riveill**, Professeur des Universités au laboratoire Informatique, Signaux et Systèmes de Sophia Antipolis (I3S), ainsi que monsieur **Pierre Tchounikine**, Professeur des Universités au Laboratoire d'Informatique de l'Université du Maine (LIUM) pour m'avoir fait l'honneur de rapporter et de juger mes travaux.

Enfin je tiens à remercier monsieur **Pierre Dillenbourg**, Professeur à l'Ecole Polytechnique Fédérale de Lausanne (EPFL) et monsieur **Olivier Walbecq**, directeur général d'Archimed d'avoir accepté de faire parti de mon jury.

Cette thèse n'aurait pas pu avoir lieu sans **Alain Derycke** qui m'a accueilli au sein de son laboratoire. Il a toujours su insuffler des directions de recherches et aplanir les difficultés extérieures me permettant ainsi de me concentrer sur mon travail de recherche. Pour tout cela, je l'en remercie.

Un grand merci à **Yvan Peter**, qui depuis mon arrivée en DEA a la lourde tâche de me supporter quotidiennement. Nos discussions, portant parfois sur notre domaine de recherches ;-), m'ont permis de mieux appréhender mon sujet et de canaliser, formaliser et argumenter toutes les idées plus ou moins " loufoques " qui m'ont traversé l'esprit ces trois dernières années. Nos déplacements communs furent toujours enrichissants aussi bien du point de vue scientifique que brassicole et humain. Les rencontres réalisées furent souvent mémorables, comme celle de Thomas Hardy :-).

Je tiens à adresser un remerciement chaleureux à **Claude Viéville** qui m'a fait découvrir le domaine de la formation à distance. Ces remarques et suggestions permirent de rester en contact avec les préoccupations des éventuels utilisateurs de COW. Grâce à ses talents de polyglotte, il m'a également enseigné le vocabulaire de survie pour voyager partout dans le monde (cerveza, birra, cerveja, ⊓NBO, . . .). J'espère que notre collaboration continuera dans le futur.

La bonne ambiance ludique de travail du bureau fut assurée par les joutes musicales avec **José Rouillard**, mon colocataire de bureau. Il m'a initié au monde *merveilleux* de l'IHM. Avant son arrivée j'utilisais `vim` dans un terminal maintenant j'utilise `vim` dans Xwindow :-).

Je voudrais également remercier tout ceux qui au détour d'un couloir ou à la machine à café, ont égayé le temps passé au labo, en remerciant plus particulièrement **Jean-Claude Tarby**, **Brigitte Bencteux**, **Florence Fournier** et **Patrick Antontchick**.

Ne vivant pas que d'amour et de bières fraîches (ou tiède si elle est noire et en provenance de Dublin), je tiens à remercier la société Archimed qui a financé ma thèse dans le cadre d'un contrat CIFRE, et plus particulièrement **Mongy Zidi** son président directeur général.

Je tiens également à remercier les enseignants qui m'ont initié à l'informatique et qui m'ont un jour donné envie de continuer un peu plus, notamment **Xavier Redon** et **Bernard Carré**. Avec une mention toute particulière pour Xavier, qui bien que connaissant mon niveau dans le domaine des réseaux informatiques a bien voulu de moi pour participer aux modules de formation dont il a la charge. Cette expérience est vraiment très enrichissante tant sur le plan technique que sur le plan humain.

Je tiens à adresser mes remerciements aux relecteurs attentifs qui tels des `gdb` orthographiques ont débuggué ce mémoire et plus particulièrement à **Marie Lorthois** et **Jacques Sleewaegen** ainsi qu'à **Sébastien Pruvôt**.

Enfin, je remercie **mes parents** qui m'ont toujours apportés les deux plus belles choses au monde, l'Amour et la Liberté.

A mes parents

« – Je ne sais pas ce que je cherche.
– Et pourquoi ?
– Parce que. . .parce que. . .il se pourrait que, le sachant, je ne sois plus capable de le chercher. . .
– Comment ? T'es dingue ou quoi ?
– Voilà une question que je n'ai pas encore réglée, observa tranquillement Zappy. Je ne connais de moi-même que ce que mon esprit est capable de discerner dans son état habituel. Et son état habituel n'est pas franchement excellent. »

Douglas Adam, Le Guide galactique.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Contexte et Problématique

Penser, c'est aller d'erreur en erreur.

Alain, *Propos sur l'éducation*

L'évolution des architectures client/serveur vers des architectures multi-niveaux avec clients légers de type navigateur web a entrainé la mutation des plates-formes fortement propriétaires d'éducation à distance (CBT : Computer Based Training) vers des plates-formes plus ouvertes reposant sur les protocoles standards de l'Internet (WBT : Web Based Training). Le principal avantage de ce changement réside dans l'accès aux informations qui désormais peut se faire à partir de n'importe quelle machine et à n'importe quel moment. Le leitmotiv de cette génération de plate-forme se résumant par « learn anything, anytime, anywhere ».

Ces changements ont accentué quelques problèmes comme la standardisation des ressources pédagogiques, i.e., comment peuvent-elles être indépendantes d'une plate-forme particulière afin d'être réutilisées. Ils soulèvent ainsi une question plus fondamentale de l'efficacité de l'e-learning [LP01]. La majorité des avancées dans les systèmes de formation est dûe à la poussée des technologies informatiques (technology push) sans se soucier obligatoirement de la pédagogie et de l'attraction de l'apprentissage [Eva01]. Or l'apprentissage ne se résume pas à accéder à des informations mais bien à un ensemble d'activités, souvent coordonnées, telles que la résolution de problèmes [Kop01].

Les récentes recherches, notamment dans le domaine du Travail Collaboratif Assisté par Ordinateur (TCAO), forcent à repenser le centre de gravité des plates-formes en le faisant migrer des technologies vers l'utilisateur qui devient l'élément fondamental du système et qui doit avoir la possibilité de modifier ce même système.

En parallèle, la Formation Ouverte et Á Distance (FOAD) se « professionnalise », faisant apparaître de nouvelles tâches à réaliser. Ces changements sont en partie provoqués par l'élargissement du domaine de la formation qui n'est plus restreint aux universités mais qui s'implante de plus en plus dans les entreprises.

Les plates-formes d'enseignement ne doivent plus se limiter à la gestion de modules et de contenus, elles doivent supporter la possibilité de créer de véritables parcours de forma-

tion complets, c'est à dire organiser et accompagner le cheminement d'un apprenant entre différents modules. L'expression et l'échange de ces parcours demandent des langages, appelés *Educational Modelling Languages* (EML), qui doivent être à leur tour standardisés, comme le furent précédemment les ressources.

Cette approche par scénarios peut se rapprocher des processus manipulés au sein des systèmes de gestion du flux de travail (Workflow). Il existe cependant peu, à notre connaissance, de tels systèmes dans les plates-formes d'apprentissage. Cette inexistence est en partie dûe à la rigidité originelle des systèmes de workflows ne permettant pas une adaptation dynamique de leurs modèles et de leurs comportements. Cette dynamicité se révèle pourtant obligatoire dans les systèmes centrés sur l'utilisateur. Le système doit s'adapter à l'usager et non l'inverse.

L'objectif de cette thèse est de proposer une plate-forme d'exécution de scénarios pédagogiques en se basant sur une approche par système de workflows. Cette réalisation passe dans un premier temps par l'étude des langages de scénarios pédagogiques et des systèmes de workflows afin notamment de réaliser des rapprochements permettant le passage d'une modélisation à l'autre. Ce point est important car les acteurs qui interviendront dans le système auront des points de vues et des approches différentes. Ensuite, nous prendrons en compte l'aspect conception centrée sur l'utilisateur pour réaliser notre plate-forme d'exécution qui devra permettre un maximum de flexibilité pour autoriser une adaptation continue au contexte. Notre objectif n'est pas de reconstruire une nouvelle plate-forme mais bien de créer un « composant technique » réalisant l'exécution de scénarios et intégrable dans les *Learning Management Systems* (LMS) existants, comme par exemple « Le Campus Virtuel » de la société Archimed. Nous avons basé notre approche sur la programmation par composants. Cette approche nous permet la construction d'applications par liens entre « briques » existantes. Elle nous libère également de la gestion des aspects « non fonctionnels » (gestion de la sécurité, des transactions, ...) pour nous consacrer au développement « métier ». Elle correspond à notre vision d'un LMS constitué de briques indépendantes spécialisées dans des domaines particuliers. Nous retrouvons cela dans le principe de co-évolution [BDT01], l'utilisateur peut construire et adapter sa plate-forme à ses besoins en choisissant les composants les plus appropriés.

Le reste de ce chapitre est organisé de la manière suivante : dans un premier temps la section 1.1 présentera le contexte initial de notre travail et son positionnement au sein du laboratoire et plus particulièrement les différentes plates-formes à l'origine de nos travaux. Le développement des modules de formation se rapproche de plus en plus du développement des logiciels informatiques. Nous présenterons dans la section 1.2 le processus de conception d'un module de formation et nous indiquerons les différents rôles qui y sont attachés. Cela nous permettra d'identifier les divers besoins et de construire notre plate-forme en respectant ces exigences. Dans la suite de ce mémoire, nous positionnerons nos travaux par rapport aux différentes étapes et aux différents acteurs impliqués. La section 1.3 exposera la problématique générale qui a guidé nos recherches. Nous terminerons enfin ce chapitre par la présentation détaillée de l'organisation de ce mémoire.

1.1 Point de départ

Cette thèse se situe dans un double contexte, à la fois un travail de recherche et un milieu industriel. Les travaux réalisés dans le cadre de cette thèse s'intègrent dans la suite des différentes plates-formes d'éducation à distance réalisées par le laboratoire TRIGONE ([Hoo95], [Bou00]). Ils reposent sur l'axe système en s'appuyant sur le projet *PLACE* (PLAtefome à Composants Evolutive). Cette thèse a également un aspect industriel dans ses applications, qui seront reprises dans les prochaines évolutions de la plate-forme « Le Campus Virtuel » de la société Archimed.

1.1.1 Le projet PLACE

1.1.1.1 Le contexte

Les applications de travail coopératif supportent trois types de fonctions : des fonctions de production qui ont trait aux objets produits par la coopération ; des fonctions de coordination qui définissent les acteurs et les tâches ainsi que les règles d'attribution des tâches aux acteurs ; enfin des fonctions de communication qui permettent l'échange d'informations entre acteurs sans interprétation par le système. N. Graham et J. Grundy ont fait une synthèse des propriétés qui doivent être supportées par un système de travail coopératif à la suite du Workshop sur les propriétés nécessaires aux outils de TCAO lors d'EHCI'98 [GG98]. Le tableau 1.1 fait une synthèse de ces propriétés.

Fonctions	Propriété
Production	Gestion flexible du degré de coopération Configuration par l'utilisateur des vues et des politiques d'accès aux données Gestion de l'historique de la coopération
Coordination	Coordination basée sur les rôles Gestion du « tour de parole » Automatisation des tâches Possibilité de gérer la visibilité de ses actions Mécanismes de vote
Communication	Communication synchrone et asynchrone Possibilité d'annotation des données Gestion de la conscience de groupe

TAB. 1.1 – Propriétés des outils de TCAO

Les trois fonctions ne sont pas remplies de manière égale par tous les logiciels de travail coopératif et, de plus, elles peuvent prendre une importance variable pour un même outil suivant l'évolution de l'activité en cours. Cette nécessité d'évolution du logiciel pour s'adapter

à l'activité est apparue comme une propriété fondamentale pour le succès des systèmes de travail coopératif.

Deux axes de recherche sont explorés dans la communauté du TCAO pour aboutir à des logiciels de travail coopératif flexibles :

- Le premier axe (et le plus ancien) repose sur l'utilisation de langages réflexifs et sur l'Implémentation Ouverte qui offrent l'opportunité de modifier le logiciel en cours d'exécution pour l'adapter à l'évolution de l'activité ;
- La deuxième approche est apparue avec le développement des technologies à base de composants. Le principe est de conserver accessible l'architecture du logiciel (composants et liens entre les composants) de manière à pouvoir la modifier par composition ou extension. On peut par exemple ajouter de nouveaux outils nécessaires à une étape de l'activité ou modifier les interactions entre ces outils pour répondre à de nouvelles règles de coordination.

Nous cherchons à combiner les avantages des deux approches, la première nous permettant d'agir finement sur le comportement et la deuxième offrant la possibilité d'ajouter facilement de nouvelles briques notamment en se basant sur des technologies et des standards qui sont largement adoptés pour le développement d'applications sur Internet.

1.1.1.2 Description du projet

Les travaux portant sur l'utilisation de composants pour la flexibilité reposent essentiellement sur la technologie JavaBeans et tirent profit des propriétés du langage Java en terme d'introspection et de chargement dynamique de classes. Toutefois, ce type de composant utilise un modèle d'interaction local et doit être étendu pour être utilisé dans un cadre réparti. Dans le domaine des applications traditionnelles on observe l'utilisation d'architectures multi-niveaux composées d'une partie présentation statique (HTML) ou dynamique (Applets ou composants JavaBeans) qui accède au niveau métier supporté par des technologies serveur (Enterprise JavaBeans) qui gèrent et manipulent les données du système d'information. Au commencement de nos travaux, ce type d'architecture n'était pas forcément employé pour la construction des systèmes de TCAO. En particulier, nous avions trouvé très peu de travaux utilisant les EJB pour la partie serveur. Or cette technologie nous semble particulièrement adaptée pour supporter l'espace de production et l'espace de coordination.

Le projet *PLAteforme à Composants Évolutive* (PLACE) se focalise sur l'utilisation des technologies de composants existantes de manière à obtenir des logiciels flexibles.

1.1.1.3 Architecture de la plate-forme

L'architecture globale est présentée à la figure 1.1. Le projet repose sur le standard Enterprise JavaBeans (EJB) et les services de la plate-forme technique Java 2 Enterprise Edition (J2EE). Ce choix se justifie par l'objet de la recherche proprement dite qui cherche à étudier l'utilisation des composants pour réaliser des environnements de travail coopératifs flexibles et d'un point de vue pragmatique par le souci de travailler sur des standards et de profiter de la richesse des services fournis par J2EE. Au niveau supérieur nous trouvons en ensemble de services fondamentaux pour la constitution d'une plate-forme de FOAD : un service de gestion

des utilisateurs et des groupes, un service de gestion des rôles et des droits associés [VP02], un service de notification et un service de workflows pour la coordination des activités. Au plus haut niveau de l'architecture nous avons les différents outils manipulés par les utilisateurs. Ces outils s'appuyent sur les services de la couche inférieure. Il est ainsi possible par exemple grâce aux services de notification de permettre à une application isolée de devenir une application coopérative avec très peu de changement dans le code. Cette modification a notamment été testée pour la conception et le développement de ArgoGraph un outil d'argumentation scientifique.

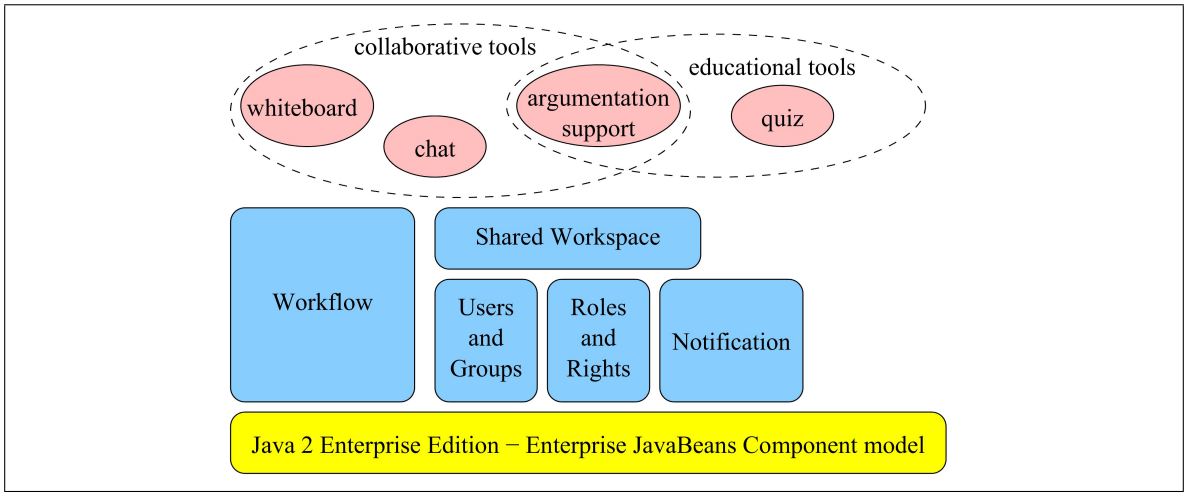


FIG. 1.1 – La plate-forme PLACE

1.1.2 La plate-forme DARE

La thèse de G. Bourguin [Bou00] propose une plate-forme de coopération fondée sur la théorie de l'activité. DARE (Distributed Activities in a Reflexive Environment) est un méta-groupware réflexif. L'objectif principal de la plate-forme est de fournir un cadre pour le support de l'activité. Elle appartient à la première catégorie d'approche pour l'introduction de la flexibilité dans le TCAO.

1.1.2.1 Présentation

L'un des fondements de DARE est d'inclure l'activité humaine au cœur des systèmes de TCAO. Le cadre théorique utilisé est celui de la théorie de l'activité, détaillé dans la sous-section suivante.

1.1.2.2 Théorie de l'activité

La théorie de l'activité est une structure très générale pour conceptualiser les activités humaines. Elle fournit une formulation de la manière dont les gens agissent dans une société

en évolution, à partir d'une perspective matérialiste. Nous n'allons pas ici la présenter complètement, nous n'en avons ni la place ni le souhait. Le lecteur intéressé pourra se référer à [BM97, Nar95].

Les cinq principes de base de la théorie de l'activité sont :

Structure hiérarchique de l'activité. L'unité d'analyse est une activité dirigée par un objet qui motive l'activité en lui donnant une direction spécifique. Les activités sont composées d'actions, dirigées par les buts à atteindre pour réaliser l'objet. Les actions sont conscientes et différentes actions peuvent être réalisées pour atteindre le même but. Les actions sont implémentées à travers des opérations automatiques. Les opérations n'ont pas leurs propres buts. Elles fournissent un ajustement des actions aux situations actuelles. Les constituants d'une activité ne sont pas fixes mais peuvent changer dynamiquement lorsque les conditions changent.

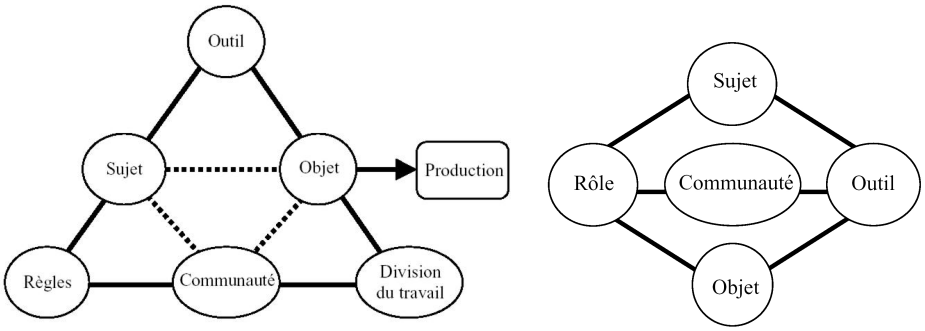
Orienté par l'objet. Le principe d'orientation par l'objet (*object-orientedness*) spécifie que les êtres humains vivent dans une réalité objective au sens large : les éléments qui constituent la réalité ne possèdent pas seulement des propriétés objectives selon les sciences naturelles mais également selon des propriétés sociales et culturelles.

Internalisation/externalisation. La théorie de l'activité différencie les activités internes et les activités externes. Elle souligne que les activités internes ne peuvent pas être analysées séparément des activités externes car elles se transforment les unes les autres. L'internalisation fournit un moyen pour essayer des interactions avec la réalité sans les réaliser avec de véritables objets (simulations mentales, imagination, réflexion sur différentes approches possibles pour résoudre un problème, *etc*). L'externalisation transforme les activités internes en activités externes. Elle est souvent nécessaire lorsqu'une collaboration entre plusieurs personnes requiert que leurs activités soient externalisées afin de pouvoir être coordonnées.

Médiation. L'activité humaine est médiatisée par des outils au sens large. Les outils sont créés et transformés pendant le développement de l'activité elle-même et ils portent en eux une culture particulière provenant de l'historique de leurs développement. Ainsi, l'utilisation d'un outil est une accumulation et une transmission d'un savoir social. Les outils influencent la nature du comportement extérieur ainsi que le fonctionnement mental des individus.

Développement. La théorie de l'activité requiert que la manière dont un humain interagit avec la réalité soit analysée à l'intérieur du contexte de développement de l'activité. Elle voit les pratiques comme des résultats d'un certain développement historique sous certaines conditions et comme un processus continu de re-formation et de développement.

La structure d'une activité dans DARE (figure 1.2(b)) s'appuie sur la structure basique de l'activité (figure 1.2(a)) défini par Engeström [Eng87]. L'*objet* (ou objectif) est la cible de l'activité. Les *outils* référent à des artefacts internes ou externes qui aident à atteindre la *production* de l'activité. La *communauté* comprend une ou plusieurs personnes qui partagent les objectifs avec le *sujet*. Les *règles* régulent les actions et les interactions à l'intérieur de l'activité. La *division du travail* définit la manière dont les tâches sont divisées horizontalement



(a) Modèle d'Engeström (b) Modèle dans DARE

FIG. 1.2 – Structure basique d'une activité (tiré de [Bou00])

entre les membres de la communauté ainsi qu'une référence aux divisions verticales du pouvoir et des statuts.

1.1.2.3 Architecture

Afin de fournir un système malléable qui puisse être adapté par l'utilisateur, DARE s'appuie sur les principes de l'*Open Implementation* définie par Kiczales [Kic96] et sur les techniques du *metaobject protocol* [KdRB91]. Chaque application (au sens large, il peut s'agir d'objet, de modules, ...) dispose de deux interfaces. La première interface correspond aux services fournis par l'application. La deuxième interface est une méta-interface qui permet d'explorer et de manipuler l'application en cours d'exécution.

L'implémentation fut réalisée en utilisant des langages offrant des capacités réflexives, Smalltalk pour la partie serveur et Java pour la partie cliente composée d'applets. La communication entre le serveur et les clients fut réalisée par l'intermédiaire du bus à objets CORBA [OMG98, GGM98].

1.1.2.4 Principaux problèmes soulevés

La gestion de la production. Un des éléments de la TA est la production, i.e. le résultat de l'activité. Le travail produit et son utilisation ultérieure ne sont pas traités dans la plateforme. Ils ne sont pas non plus gérés (stockage, indexation dans un référentiel, *etc*).

La gestion du temps. Dans le cadre d'applications « réelles » la production doit généralement se faire dans un temps fixé, reflétant des conditions « industrielles ». Cette contrainte n'est pas gérée de manière explicite dans DARE. Les acteurs doivent eux-même décider de mettre fin à leurs activités.

La coordination explicite. Une activité existe rarement en complète isolement, i.e., elle a souvent des liens avec d'autres activités. Ces liens sont de plusieurs types. Il peut s'agir de liens d'ordonnancement, de liens de production, de liens temporels, *etc.* Cette coordination explicite entre activités est laissée de côté par la plate-forme DARE actuelle. La coordination est obtenue de manière implicite par une notion producteur/consommateur. Certaines activités ne peuvent commencer que si la production d'une autre activité est disponible.

1.1.3 Le Campus Virtuel ©

La plate-forme dite « Le Campus Virtuel » fait suite aux travaux de thèse de Frédéric Hoogstoel [Hoo95] et à ses améliorations lors de projets européens (Modem [Vié98], Demos [VD97]). En 1999, avec l'appui de l'ANVAR, cette plate-forme fut l'objet d'un transfert industriel avec la société Archimed (SSII de la métropole Lilloise) [Arc03a]. La version industrielle comprend quatres services principaux (inscription, travail de groupe, organisation du travail et delivery) articulés autour d'un bureau virtuel [Vié02] (voir figure 1.3). Ce dernier permet d'accéder aux différents services. Il est accessible via un navigateur internet et est personnalisable par l'utilisateur.

Le service d'inscription permet d'accéder à la liste des modules de cours disponibles. Un module est composé d'un ensemble prédéfini d'activités pédagogiques. Ce service offre la possibilité aux étudiants de s'inscrire dans un ou plusieurs modules. La validation de l'inscription peut être manuelle ou automatique.

Le service de travail de groupe comprend les services de communication (courrier électronique et forums de discussions) et un système de gestion des fichiers permettant aux étudiants et aux enseignants de disposer de casiers pour y déposer leurs documents. En plus du casier personnel, il est possible de créer des casiers pour des groupes. Par exemple, un enseignant peut placer un texte à étudier dans le casier du groupe dont il a la charge et prévenir les étudiants par l'envoi d'un courrier électronique.

Le service organisation du travail permet aux étudiants de structurer leurs tâches et aux enseignants de suivre l'avancement des étudiants. Il est constitué de deux composants principaux, un système d'agenda électronique et un moteur de workflows. L'agenda permet aux étudiants d'y mettre tous leurs rendez-vous comme dans un classique agenda papier mais ils trouveront également dans celui-ci les informations concernant les différentes activités pédagogiques dans lesquelles ils sont inscrits (date de début et de fin, travail à rendre, ...). Cela permet aussi aux enseignants de fixer des rendez-vous pour des examens ou des activités synchrones présentielles ou à distance. Le moteur de workflows permet de structurer et d'organiser les différentes activités pédagogiques d'un module d'enseignement. Ce service offre notamment aux étudiants une vision globale de leurs parcours et facilite le suivi des enseignants.

Le service de référentiel indexe et stocke les différentes ressources pédagogiques utilisables dans les activités. Il s'appuye sur le standard de catalogage *Learning Object Metadata* (LOM). Il permet ainsi aux enseignants de trouver rapidement et facilement des ressources en fonction de leurs besoins.

1.1.3.1 Particularité : un système de workflows pour la gestion des formations

La principale particularité de cette plate-forme est l'incorporation d'un moteur de workflows pour gérer l'avancement des modules de formations, faisant ainsi du Campus Virtuel l'une des premières plates-formes d'enseignement à distance qui ne s'intéresse pas uniquement à la fourniture de contenus mais plutôt à la mise en place de scénarios pédagogiques. Nous définirons de manière précise les systèmes de workflows dans le chapitre 3. Pour l'instant, nous pouvons les définir comme des systèmes gérant la coordination des tâches, i.e., déterminant un ordre d'exécution des tâches et la circulation d'informations entre ces tâches. Dans le cadre de l'éducation à distance, cela peut correspondre à l'enchaînement des tâches d'un module d'enseignement (faire le cours **puis** faire les exercices, ...) ou l'enchaînement et le positionnement des modules les uns par rapport aux autres (d'abord le module d'inscription, puis en parallèle le module de math et le module de français, ...). Dans « Le Campus Virtuel » cela permet au formateur/tuteur de mettre de côté l'aspect coordination pour se concentrer sur le suivi des apprenants.

1.1.3.2 Architecture

Le Campus Virtuel s'appuie sur deux briques de bases SIM et MASC. L'architecture est représentée à la figure 1.3.

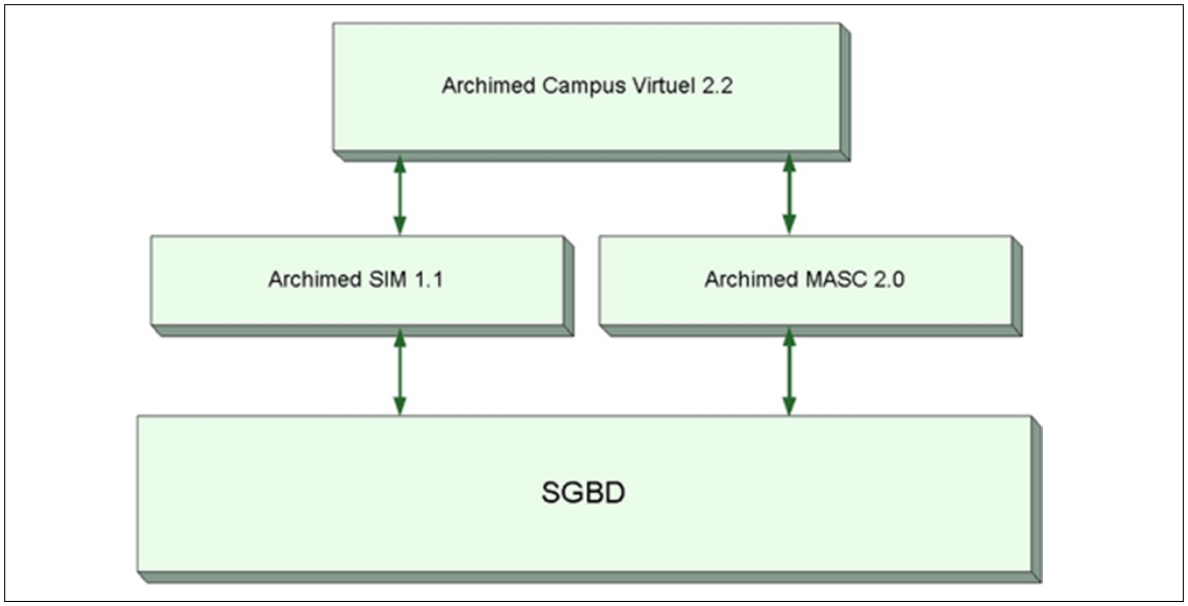


FIG. 1.3 – Architecture du Campus Virtuel

SIM (Système d'Information Multimédia) est un progiciel de GED (Gestion Électronique de Documents) qui permet d'organiser, rechercher et diffuser tout type d'informations (données, documents bureautiques, images, sons, vidéo, *etc*). Dans le cadre du Campus, il est utilisé

pour la gestion, la persistance et l'indexation des données, comme par exemple les ressources pédagogiques et les modèles de parcours. Il contient également le moteur de workflows qui gère la circulation des différents documents entre les participants.

MASC est une solution de gestion de portails capable d'héberger un ensemble de services accessibles en Intranet, Extranet ou via Internet. Dans le cadre du Campus, il est utilisé pour la sécurisation et la gestion des profils utilisateurs notamment pour la personnalisation du bureau virtuel.

1.1.3.3 Principaux problèmes soulevés

L'utilisation d'un moteur de workflows pour la gestion et la planification des modules d'enseignement est un élément innovant de la plate-forme ; cependant, il fut conçu à l'origine pour de la gestion électronique de documents (circulation de documents administratifs par exemple). Il ne prend donc pas en compte les particularités inhérentes au domaine de la formation à distance telle que la gestion des groupes et les parcours individualisés.

L'autre point important pour un moteur de workflows est sa flexibilité, i.e. la possibilité de redéfinir dynamiquement le parcours de formation lors de son exécution pour l'adapter au contexte en prenant mieux en compte les besoins des utilisateurs comme par exemple ajouter des exercices afin de mieux faire comprendre un cours. Il peut également être difficile pour un enseignant de modéliser de manière complète son module d'enseignement. Afin de résoudre ce problème, il est nécessaire de laisser des zones de flous qui seront peu à peu comblées lors de l'exécution du module.

D'un point de vue industriel, il serait intéressant que le moteur respecte les différents standards existants notamment pour ce qui concerne la description des processus [Wor99a] et l'interopérabilité avec d'autres moteurs [Wor01]. Cela peut s'avérer intéressant si l'on souhaite faire collaborer entre elles des plates-formes d'enseignements hétérogènes gérées par des organismes de formations différents.

1.1.4 Positionnement de nos travaux

Nos travaux se situent à la croisée des chemins des trois plates-formes précédentes. D'un point de vue technologique, nous nous situons pleinement dans le projet PLACE en constituant une des briques principales. Nous travaillons principalement sur l'aspect coordination des tâches à réaliser, ce qui nous rapproche du Campus Virtuel. Nous pensons qu'un système de workflows est une solution envisageable pour la gestion du cheminement d'un étudiant aussi bien entre les modules de formation qu'à l'intérieur de ceux-ci pour l'ordonnancement explicite des activités. Ce qui nous rapproche de DARE et de sa conception basée sur la théorie de l'activité. Une approche par workflow nous permettrait de gérer la production des activités en assurant sa circulation et offrirait une solution concernant l'aspect temporel qui est fondamental dans de tels systèmes.

Après la présentation de l'origine et du positionnement de nos travaux, nous allons maintenant nous intéresser aux processus de conception et aux différents rôles afférents dans le cadre d'une plate-forme de formation à distance.

1.2 Processus de conception, d'utilisation et rôles associés

L'architecture des plates-formes de FOAD est d'une complexité grandissante. La conception puis la mise à disposition et l'utilisation des modules d'enseignement nécessitent un grand nombre de tâches qui demandent une spécialisation accrue des différents acteurs qui interviennent tout au long du cycle de vie. Cette division du travail est accentuée par l'industrialisation de la FOAD et sa mise en œuvre de plus en plus importante au sein des entreprises.

Cette section présentera successivement le processus de conception, le processus d'utilisation et les différents rôles intervenants dans ces processus.

1.2.1 Processus de conception

Le processus de conception d'un module d'enseignement, visible à la figure 1.4, comprend 6 phases principales. Nous pouvons le rapprocher du *processus unifié de développement* (*Rational Unified Process* : RUP)[JBR00]. Chaque phase permet de passer à la suivante et également de faire un retour en arrière lorsque des problèmes inhérents à la phase précédente apparaissent.

Expressions initiales des besoins et des modèles. Cette phase correspond à l'expression initiale d'un module de formation. Un *enseignant* va décrire de manière informelle (non compréhensible directement par un système informatique) les différentes activités du module en fixant des objectifs globaux ou locaux et éventuellement des prérequis pour réaliser le module. Il précisera également le type de contenu et les ressources pédagogiques qui seront nécessaires pour l'accomplissement du module. Dans le RUP, cette phase correspond à l'élaboration des cas d'utilisation (*use case*) définis en UML [Mul99]. Ces cas d'utilisation présentent l'avantage d'être compréhensibles facilement par l'ensemble des acteurs du processus de développement logiciel. Il n'existe pas à ce jour de formalisme équivalent spécialisé dans le domaine de la FOAD pour décrire le comportement des différents acteurs et des activités dans le cadre d'un module d'enseignement.

Analyse et conception. A la suite de la description informelle, un *ingénieur pédagogique* qui connait les particularités de la plate-forme de destination, en collaboration avec l'enseignant formalise le scénario pédagogique en le découpant en différentes phases reliées entre elles. L'expression de ce modèle sera réalisée en utilisant un langage de modélisation pédagogique. Au cours de cette phase, certaines activités pourront être remises en causes car elles nécessitent des caractéristiques que n'offre pas la plate-forme de destination. Ce travail d'analyse et de conception fera également appel aux *fournisseurs de ressources pédagogiques* et aux *développeurs de composants*. Les premiers

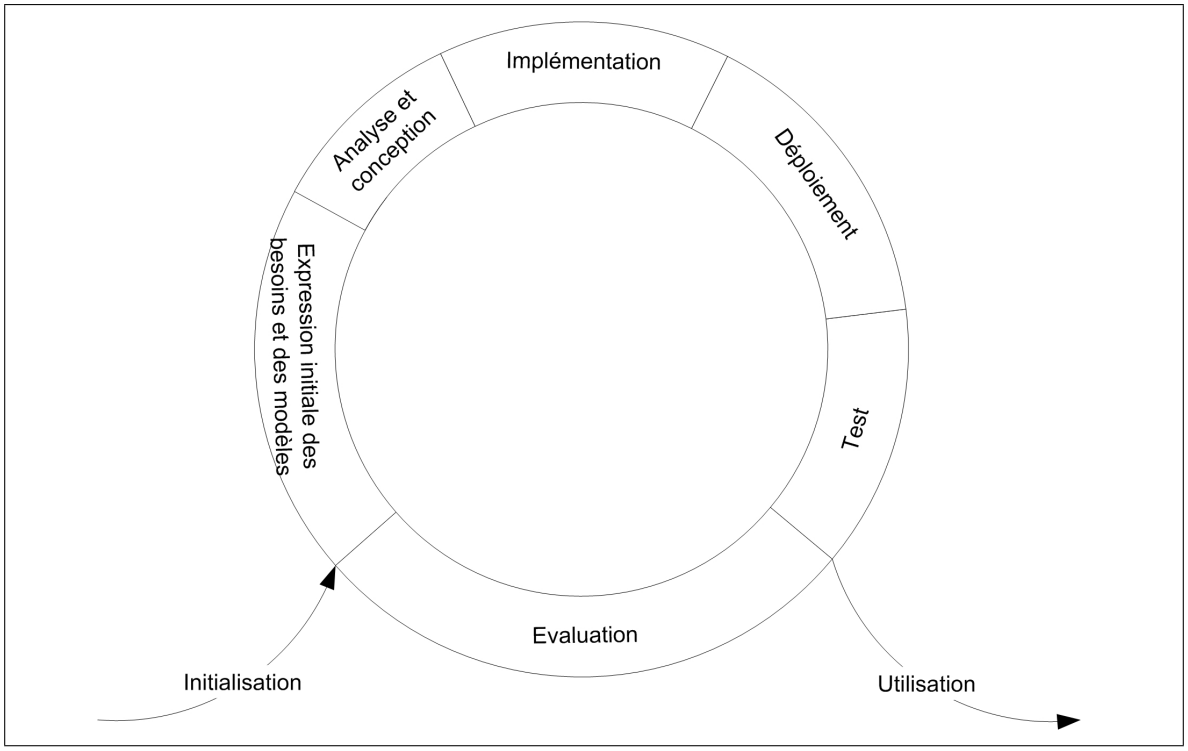


FIG. 1.4 – Processus de développement d'un module de formation

indiqueront les ressources disponibles ou à développer pour répondre aux besoins de l'enseignant et les deuxièmes pourront définir les fonctionnalités à développer pour permettre à la plate-forme de supporter complétement le module.

Implémentation. Cette phase consiste à réaliser l'ensemble des composants techniques et métiers (ressources pédagogiques) nécessaires pour le module d'enseignement. Le rôle principal revient ici au développeur de composants. Il va collaborer avec l'ingénieur pédagogique et avec le fournisseur de contenu pour la définition des composants à développer et éventuellement avec *l'assembleur de composants* pour la modification des assemblages existants ou pour la constitution de nouveaux composants.

Déploiement. Tous les éléments développés ou récupérés dans les phases précédentes sont mis en place sur la plate-forme. Cette phase concerne l'assembleur de composants et *l'administrateur de plate-forme*. A la fin de cette phase, le module peut alors être référencé.

Test. Cette phase permet la vérification du comportement de la plate-forme, la cohérence des modèles et la bonne agrégation de tous les composants de la plate-forme. Ces tests seront réalisés par l'ingénieur pédagogique qui assurera de cette manière la validité d'un point de vue pédagogique et fonctionnel. Le développeur de composants et l'assembleur de composants contrôleront l'absence de bugs et le bon fonctionnement de l'ensemble technique.

Évaluation. Au cours de l'utilisation des modules d'enseignement, il est possible d'évaluer le comportement des apprenants et des enseignants afin de vérifier que le modèle prévu initialement correspond bien aux besoins des différents acteurs et éventuellement détecter les ajustements récurrents qui pourront par la suite être directement intégrés dans le module et ceci dans une démarche d'amélioration continue. Le comportement de la plate-forme au cours de son utilisation en condition réelle peut également être évalué afin de déterminer s'il existe des problèmes de surchage des serveurs, afin d'y apporter des solutions en reconfigurant et en redéveloppant certains composants de la plate-forme. Cette tâche concerne principalement les *administrateurs de la plate-forme* qui pourront ainsi faire des recommandations aux ingénieurs pédagogiques et aux développeurs de composants.

La phase de **constitution des parcours de formation** se trouve à la limite entre le processus de conception et celui d'utilisation. Dans cette phase, l'ingénieur pédagogique avec l'aide des enseignants de divers modules et du conseiller en formation, peut réaliser des parcours de formations, i.e., l'assemblage de différents modules pour constituer par exemple une année de formation complète. Cette création se décompose selon les mêmes phases que le module de conception.

1.2.2 Processus d'utilisation

Le processus d'utilisation détermine les différentes phases relatives à l'exécution d'un module de formation.

Instantiation. La première phase consiste à déterminer les différentes personnes qui prendront part à la formation [1] ainsi que leurs rôles (*apprenant*, *tuteur*, *référent*). La constitution des groupes d'apprenants associés à la formation pourra se faire avec l'aide du *conseiller en formation*. Le tuteur vérifiera la présence des différentes ressources nécessaires pour débuter. La formation pourra ensuite commencer.

Execution de la formation. Nous sommes ici en présence de la phase visible de l'iceberg. C'est à partir de ce moment que la plate-forme va réellement être utilisée. L'exécution « normale » consistera en l'enchaînement des différents modules et de leurs activités respectives. Un tuteur assure le contrôle du bon déroulement d'un module. Un référent représente la personne responsable d'un groupe de formation. Lors de l'exécution différents problèmes d'ordre technique, pédagogique ou administratif peuvent être rencontrés. Les premiers seront principalement résolus par l'administrateur de la plate-forme (plantage du système d'exploitation, coupure réseau, défaillance matérielle). Les seconds sont les plus nombreux et les plus difficiles à résoudre. Le tuteur doit par exemple faire face au problème des compétences et du niveau des apprenants afin d'adapter le contenu du module, voire son organisation comme par exemple redéfinir l'ordre des différentes activités. Ce problème s'accentue lorsque les modifications ne s'adressent pas au groupe

[1] Nous utilisons ici le terme formation pour faire à la fois référence à un module d'enseignement ou à un parcours de formation.

complet mais à une partie des membres. Le troisième type de problème relatif aux tracas administratifs (problème d'inscription, de paiement, ...) pourront être réglés par le responsable administratif.

1.2.3 Rôles liés à une plate-forme de FOAD

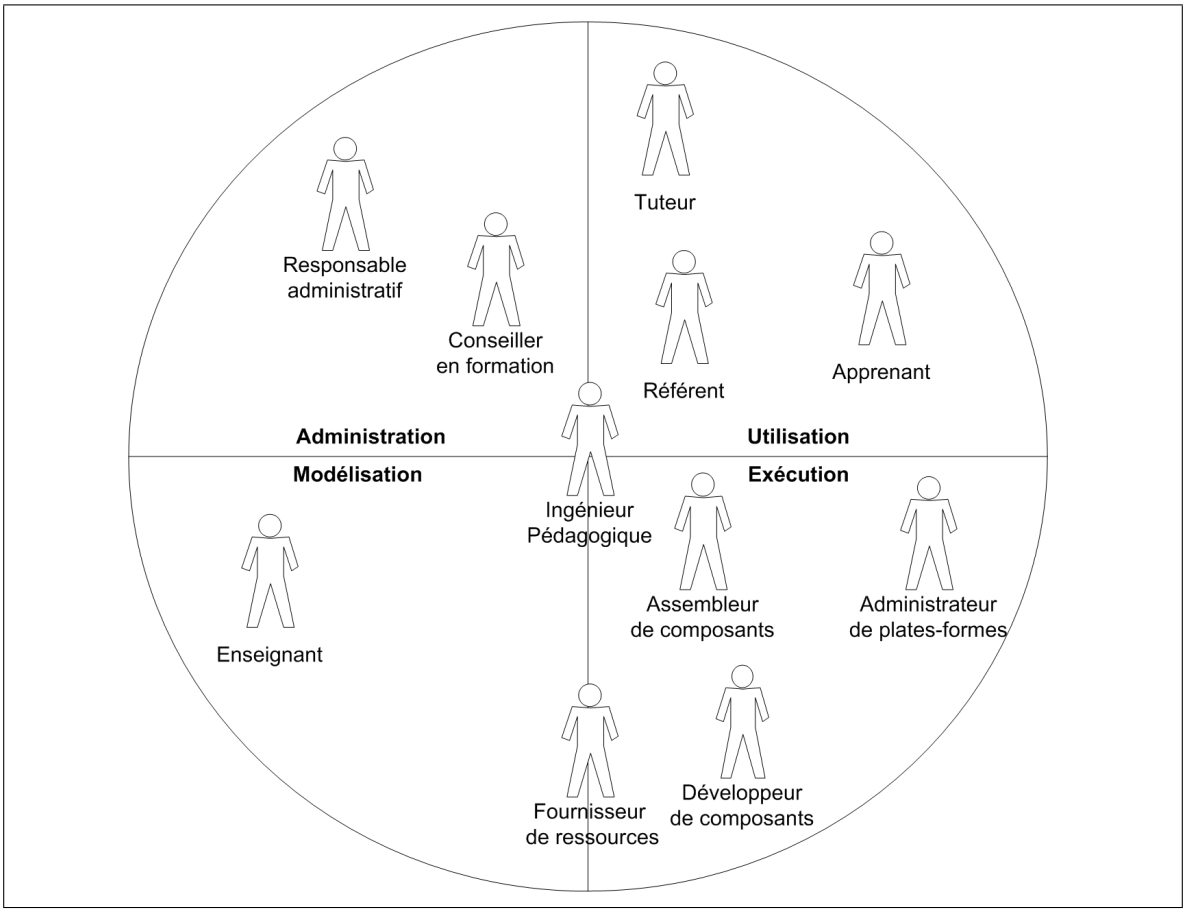


FIG. 1.5 – Domaines fonctionnels liés à une plate-forme de FOAD

Afin de préciser les différentes tâches, nous supposons qu'elles sont reliées à des rôles particuliers les réalisant. Notre division en rôles (figure 1.5) ne cherche pas à être exhaustive. Elle s'appuie sur notre expérience et nos recherches dans le domaines des EIAH (Environnement Informatique pour l'Apprentissage Humain). Dans les chapitres suivants de ce mémoire, nous ferons souvent référence à ces rôles afin d'indiquer vers qui se sont portés nos efforts. Cette subdivision en rôles permet de cerner les besoins et les attentes des différentes personnes intervenants dans le cycle de vie d'une plate-forme d'enseignement. Ce type de découpage est également utilisé dans le domaine du génie logiciel afin de déterminer les tâches à réaliser lors du processus de conception [JBR00] et de développement d'applications [MS01, SW03]. Nous

nous intéresserons ici aussi bien aux rôles des utilisateurs finaux qu'à ceux des concepteurs et développeurs de solutions pour la FOAD.

Afin de mieux catégoriser les différents rôles, nous avons commencé par distinguer les différents domaines fonctionnels qui sont :

- les tâches administratives (gestion des inscriptions et des compétences, ...) ;
- les tâches de modélisation des modules et des parcours de formation ;
- les tâches techniques, c'est à dire le développement et la maintenance de la plate-forme ;
- les tâches d'utilisation de la plate-forme.

Nous avons ainsi obtenu 11 rôles différents que nous avons introduits lors de la description des processus de conception et d'utilisation. Même si une personne peut tenir plusieurs rôles, la structure des plates-formes actuelles (notamment leur taille) empêche une même personne de tenir la totalité des rôles présentés.

1.2.3.1 Relations entre les rôles

Comme nous l'avons évoqué dans les sections précédentes, les rôles interagissent entre eux. Nous avons représenté toutes ces relations dans la figure 1.6.

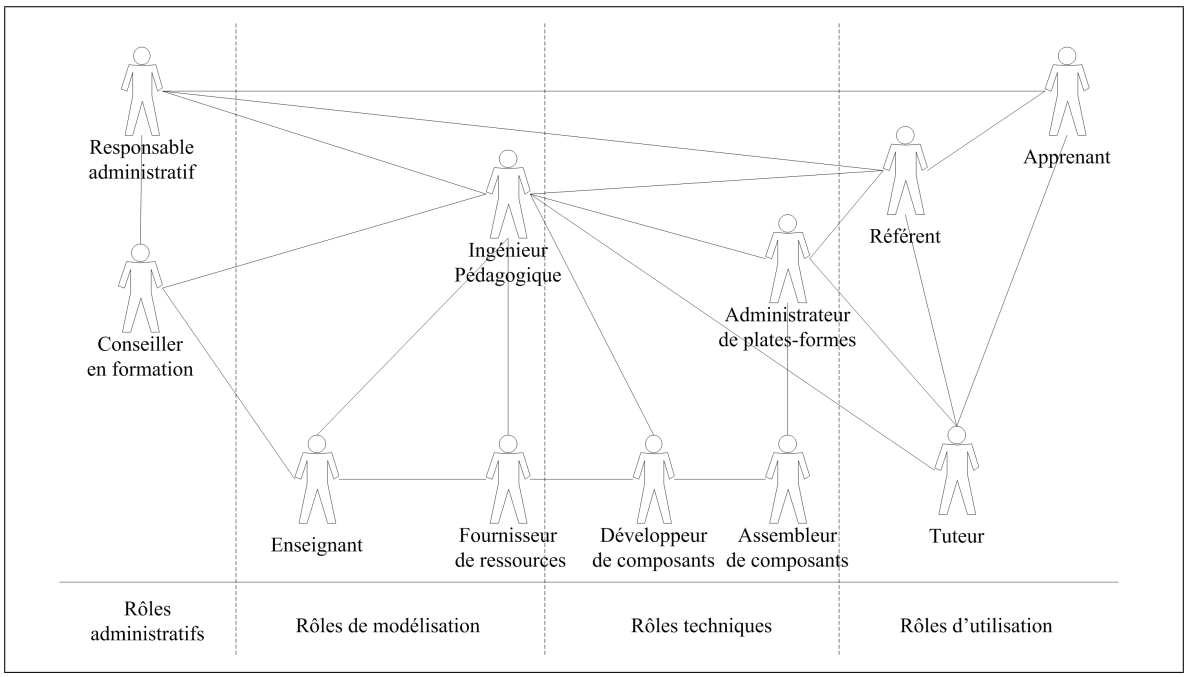


FIG. 1.6 – Diagramme de collaboration des rôles

Nous allons maintenant présenter plus en détail les différents rôles que nous avons identifiés.

1.2.3.2 Enseignant

L'enseignant constitue le premier maillon de la chaîne de création d'un module pédagogique ou d'un parcours de formation. Il définit, en terme de compétences, les prérequis et les objectifs des différentes activités pédagogiques. Il associe éventuellement des services (systèmes de discussion synchrones ou asynchrones), des outils spécifiques (éditeurs de textes, simulateur d'oscilloscope, ...) et des ressources pédagogiques (cours de physique d'électricité et questionnaires associés par exemple). Des zones de flou peuvent exister dans la déclaration des activités car il est souvent difficile de prévoir entièrement le déroulement d'un module. L'enseignant ne possède pas obligatoirement une vue globale du processus d'apprentissage. On peut le caractériser comme expert d'un domaine particulier. Il s'exprime de manière informelle, c'est à dire dans un langage non compréhensible directement par un système informatique. Cette description peut s'exprimer à la manière d'un scénario ou d'une pièce de théatre en décrivant toutes les scènes (rôles, actions à réaliser, ...) et leurs enchaînements. Il doit tenir compte du contexte (de la plate-forme et administratif) pour éventuellement modifier et ajuster sa description. Il collabore avec l'ingénieur pédagogique qui maîtrise les fonctionnalités existantes et envisageables de la plate-forme, lors de la formalisation des modèles et de la mise en œuvre des parcours. Il fait office de consultant auprès du conseiller en formation pour juger de l'aptitude d'un éventuel apprenant à suivre la formation.

1.2.3.3 Fournisseur de ressources pédagogiques

Il conçoit, développe (ou fait développer) et met à disposition de l'enseignant et de l'ingénieur pédagogique des ressources pédagogiques (numérique ou non) pouvant être utilisées lors de l'exécution d'un module. Il représente une force de proposition lors de la formalisation des modèles, il peut les orienter en fonction du contenu pédagogique et de son ordonnancement défini indépendamment des tâches d'apprentissage. Il procure les composants « métier », i.e., les différentes briques directement liées à la fonction de base de la plate-forme, l'enseignement.

1.2.3.4 Ingénieur Pédagogique

L'ingénieur pédagogique est l'homme-orchestre de la plate-forme. Il a une vue métier sur la plate-forme. Il intervient tout au long du cycle de conception. Il fait le lien entre la modélisation des modules, leurs développements et leurs utilisations. Dans les phases d'analyse et de conception, il assure la formalisation des modèles en accord avec l'enseignant et en tenant compte des caractéristiques de la plate-forme. Il est capable de discuter avec le développeur de composants afin d'établir le cahier des charges des nouveaux éléments à introduire dans la plate-forme. Il peut guider le conseiller en formation lorsqu'il souhaite mettre en œuvre une nouvelle formation. Il peut dialoguer avec le référent et le tuteur pour réaliser des modifications du module. Ces modifications peuvent porter sur les ressources pédagogiques ou sur l'enchaînement des activités.

Ses principales exigences concernent les modèles. Ces derniers doivent prendre en compte un vocabulaire et des concepts proches de ceux de l'enseignant afin que l'ingénieur pédagogique réalise facilement, i.e., de la façon la plus naturelle possible, le passage vers la formalisation.

Cela nécessite notamment des outils de manipulation supportant une visualisation des modèles compréhensible par l'enseignant. L'ingénieur pédagogique désire également une certaine indépendance des modèles vis-à-vis de la plate-forme d'exécution. En effet l'évolution du système ne doit pas remettre complètement en cause des modèles opérationnels déjà éprouvés.

1.2.3.5 Apprenant

L'apprenant est l'utilisateur principal d'une plate-forme de FOAD. Sans lui, la plate-forme n'a pas de raison d'exister. Il utilise le système pour acquérir de nouvelles compétences. Il peut travailler seul ou en groupe.

Comme il est le rôle principal, de nombreux services lui sont nécessaires. Il a besoin d'un catalogue des enseignements notamment en fonction de ses compétences acquises ou de celles qu'il souhaite acquérir. Il peut avoir besoin d'aide pour la création et la modification de son parcours. Cette aide peut provenir du tuteur, de l'ingénieur pédagogique ou du conseiller en formation. Afin de s'intégrer et d'échanger avec les autres utilisateurs de la plate-forme, il lui faut des outils de communications asynchrones (mail, forum) ou synchrones (chat) ainsi que des outils de travail en groupe (éditeur partagé, tableau blanc, ...).

L'apprenant a besoin de connaître le travail à faire, déjà effectué et celui qui arrivera dans le futur. Cela lui permet d'avoir une vision globale du module/parcours et ainsi de mieux se situer dans le parcours global ainsi que dans les actions qu'il effectue.

D'un point de vue plus technique, il attend que la plate-forme offre un support de la mobilité (accès de n'importe quel endroit avec n'importe quel périphérique et à n'importe quel moment à la plate-forme). Il souhaite qu'on lui fournisse les outils adéquats et éventuellement qu'il puisse les personnaliser en fonction de ses besoins et de ses compétences.

1.2.3.6 Tuteur

Le tuteur a un rôle primordial pour le bon déroulement d'un module de formation dont il réalise l'animation. Il assure le suivi des différents étudiants en les guidant et en les aidant pour atteindre les divers objectifs pédagogiques. Il informe le référent du déroulement et le prévient si un étudiant présente des difficultés particulières. Le tuteur peut, en collaboration avec l'ingénieur pédagogique et le référent, apporter des modifications temporaires (i.e. liées à une instance particulière du module) pour un étudiant ou pour tous les étudiants ainsi qu'une modification définitive avec l'accord de l'enseignant. En effet, le tuteur n'est pas obligatoirement un expert du domaine. Il peut ne disposer que des connaissances suffisantes pour répondre aux attentes des étudiants dans un certain nombre de modules.

Afin de réaliser ses tâches, ce rôle nécessite des outils de communication classiques (mail, forum) pour échanger avec les apprenants. Pour faciliter le suivi, un ensemble d'outil de gestion de l'avancement ainsi que de la modification du module doivent être disponibles.

1.2.3.7 Référent

Le référent gère un ou plusieurs groupes de formation. Il permet aux apprenants d'avoir un interlocuteur unique tout au long de leurs parcours de formation. Il peut quand cela devient

nécessaire jouer le rôle de médiateur avec les différents tuteurs de modules. Il peut demander des modifications de parcours aux tuteurs.

Afin de remplir pleinement son rôle, le référent dispose d'outils semblables à ceux du tuteur comme les outils de communication et de suivi des apprenants. Il dispose en plus de systèmes permettant d'assurer le suivi sur l'ensemble du parcours des étudiants et notamment des outils de suivi de groupes. Ceci afin d'être capable de déterminer rapidement si les étudiants avancent au même rythme ou s'il y a des écarts très importants entre les plus rapides et les plus lents, et ainsi prendre les bonnes décisions pour harmoniser le travail des apprenants.

1.2.3.8 Responsable administratif

Ses tâches principales sont la gestion des inscriptions ainsi que le suivi administratif (vérification de présence, ...) des différents participants. Il est le garant du bon déroulement des formations face aux différentes autorités (ministères, entreprises, ...). Pour réaliser ces différentes tâches il doit disposer d'outils de suivi des utilisateurs.

Le responsable administratif doit pouvoir vérifier qu'une personne inscrite dans la formation réalise bien le travail demandé afin, par exemple, de pouvoir fournir des certificats de scolarités et/ou des justificatifs pour les employeurs. La notion temporelle est également importante pour le monde industriel afin de valider le temps effectif passé par l'employé sur la plate-forme.

1.2.3.9 Conseiller en formation

Le conseiller en formation accueille les apprenants et les dirige vers les cursus en fonction de leurs compétences et de leurs souhaits. Il peut, avec l'aide de l'ingénieur pédagogique créer de nouveaux parcours à partir des modules existants.

Le conseiller en formation doit avoir accès aux différentes formations et modules disponibles. Cela implique que la plate-forme dispose d'un service d'indexation.

1.2.3.10 Assembleur de composants

Le rôle d'assembleur de composants est un des trois rôles techniques. Son objectif principal est d'assembler les différents composants. Nous parlons ici de composants au sens large. Il peut s'agir de composants techniques ou de composants métiers. Une définition plus fine des rôles diviserait l'assembleur en deux, une partie dédiée aux aspects techniques et une partie dédiée aux ressources pédagogiques.

L'assembleur de composants n'a pas de besoin spécifique vis-à-vis de la plate-forme d'enseignement à distance. Elle n'est pour lui qu'un domaine d'application parmi d'autres.

1.2.3.11 Développeur de composants

Il conçoit et développe des composants logiciels en fonction des besoins exprimés par l'ingénieur pédagogique, le fournisseur de ressources et l'assembleur de composants. Il crée de nouvelles fonctionnalités pour la plate-forme.

Il définit et réalise les composants « techniques ». Il intervient principalement dans les phases de conception et d'implémentation.

Le développeur n'a pas de besoin spécifique vis-à-vis de la plate-forme d'enseignement à distance. Elle n'est pour lui qu'un domaine d'application parmi d'autres.

1.2.3.12 Administrateur de la plate-forme

Le rôle d'administrateur de la plate-forme intervient principalement au cours de l'utilisation du système par les différents utilisateurs. Il peut également intervenir lors de la phase de test afin de valider le comportement de la plate-forme. Il assure le bon fonctionnement de la plate-forme d'un point de vue technique.

Afin de réaliser ses différentes tâches, il doit disposer d'outils d'administration et de surveillance aussi bien de la plate-forme elle-même que du système logiciel et matériel sur lesquels elle s'appuie (système d'exploitation, réseau, matériel, ...). Il posséde une vision globale du système. Il peut intervenir pour régler des problèmes liés à l'accès aux différents services, il réalise les configurations par défaut des environnements, il a un pouvoir d'administration des différents comptes des utilisateurs (création, destruction, gestion des droits d'accès) en accord avec le responsable administratif (apprenants à jour de leurs inscriptions, mise à jour de la liste des tuteurs, gestion des enseignants), l'ingénieur pédagogique et le référent (pour la constitution des groupes par exemples). Il assure également les différentes sauvegardes et la remise en état après incident (coupure électrique, défaillance matérielle, incendie, ...).

1.3 Problématique générale

Comme nous avons pu le montrer dans les sections précédentes, DARE et « Le Campus Virtuel » s'intéressent à « l'activité » de manières différentes. Dans DARE, elle est l'élément central à la base du système lui-même. Elle évolue continuellement. Le Campus Virtuel se focalise plus sur la coordination des activités qui ne sont que des tâches à réaliser et qui n'évoluent pas. Il est centré sur le cheminement qui permet de passer d'une activité à l'autre et cela dans le domaine particulier de l'éducation à distance. Ces deux points de vues ne sont pas antagonistes car ils abordent chacun un aspect particulier de l'activité.

Notre travail, qui s'appuie sur les concepts des deux plates-formes précédentes, se situe dans le domaine des Environnements Informatiques pour l'Apprentissage Humain (EIAH). Notre objectif global est la fourniture d'un support à l'exécution de scénarios pédagogiques répondants à la fois aux besoins des enseignants et des apprenants.

Les principaux enjeux pour un enseignant sont de pouvoir modéliser un module d'enseignement ou un enchaînement de modules en utilisant des concepts qui lui sont familiers et qui lui permettent d'utiliser différents styles pédagogiques (constructiviste, behavioriste, ...). Ces scénarios peuvent également contenir des zones peu définies car il peut être difficile de prévoir avec précision l'intégralité d'un module. Une fois ces modèles réalisés, ils peuvent être instantiés de nombreuses fois pour des groupes d'étudiants différents. Les modèles doivent en effet être suffisamment abstraits pour être réutilisés. L'enseignant doit pouvoir suivre le déroulement du module à la fois de manière globale et de manière locale. Un suivi global consiste

à suivre l'avancement du groupe dans son ensemble et à connaître les apprenants s'écartant de façon importante de la « moyenne ». Un suivi local permet d'observer avec précision l'état d'avancement de chaque étudiant de manière individuelle. L'enseignant peut analyser le travail déjà effectué pour détecter d'éventuelles situations qui pourraient être à l'origine du retard ou de l'avance de l'étudiant par rapport aux autres membres du groupe. Nous arrivons ainsi à l'enjeu suivant pour l'enseignant, la capacité à adapter et à modifier son module d'enseignement pour prendre en compte les difficultés, les intérêts et les situations exceptionnelles d'un ou plusieurs étudiants. Les modifications peuvent alors correspondre à une redéfinition de l'activité, à un ajout d'activité ou à un réordonnancement de l'exécution.

Du point de vue de l'étudiant les enjeux sont la possibilité d'avancer à son rythme et ce même s'il appartient à un groupe. Il doit pouvoir différencier son parcours au sein du groupe. Cela nous amène à l'enjeu suivant qui est la possibilité de construire dynamiquement son parcours de formation en fonction de ses besoins et de ses attentes en puisant dans la liste des modules disponibles. Ce comportement préfigure notamment la formation tout au long de la vie et nécessite une gestion des acquis et des compétences de l'apprenant. Tous ces enjeux tournent particulièrement autour de la notion de scénario et de parcours. Pour répondre au mieux à ces besoins, nous avons choisi une approche basée sur les systèmes de workflows, i.e. des systèmes gérant le flux de travail et l'ordonnancement entre plusieurs tâches. Cette approche a été peu utilisée dans les EIAH notamment à cause de la rigidité originelle de ces systèmes qui s'accordait fort mal avec la dynamicité des activités humaines. L'autre problème que nous avons cherché à résoudre est de tisser des liens entre la vision pédagogique de l'enseignant et la vision procédurière du concepteur de workflows. Notre premier travail consistera à comprendre, analyser et rapprocher ces deux points de vues, afin de réaliser un passage d'une expression de modèles orientée métier à une expression technique. Notre deuxième travail consistera à procurer aux utilisateurs de notre système des moyens d'analyser un modèle en cours d'exécution et à apporter des solutions pour sa modification dynamique sans arrêter son déroulement. Les transformations peuvent concerner aussi bien le modèle que le comportement de la plate-forme dans sa manière de traiter le modèle.

Afin de répondre à ces enjeux de flexiblité et d'adaptation des modèles en cours d'exécution, nous réaliserons notre plate-forme en nous appuyant sur les technologies à composants logiciels qui permettent une grande souplesse de conception et nous baserons notre implémentation sur les différents patrons de conception existants. Nous tiendrons compte également d'enjeux plus généraux liés aux utilisateurs finaux comme le nomadisme et l'accès multi-canal à notre système.

1.4 Plan du document

Ce mémoire se compose de deux parties. La première présente notre problématique et les deux domaines dans lesquels nous avons travaillé (les plates-formes de FOAD et les systèmes de workflows). La deuxième partie présente COW, la plate-forme que nous avons conçue.

Dans le chapitre 2, nous nous intéresserons aux langages métiers dédiés au monde de l'éducation. Ces langages sont généralement regroupés sous la dénomination EML (Educatio-

nal Modelling Languages). Nous verrons notamment que ce terme d'EML désigne une grande variété de langages aux fonctions et aux objectifs bien différents. Cette connaissance des langages métiers nous permettra de mieux appréhender le domaine de la modélisation éducative et de déterminer quel langage existant répond le mieux à nos besoins. Notre objectif dans cette thèse n'est pas de redéfinir complètement un nouvel EML mais d'en réutiliser un existant, même si nous pouvons être amené à faire des propositions pour l'ajout d'éléments permettant une meilleure gestion par un moteur de workflows.

Comme annoncé dans la problématique, nous souhaitons baser notre démarche sur un système de coordination explicite (et parfois implicite) des tâches et des activités. Dans le chapitre 3 nous présenterons donc cette catégorie de systèmes en pleine évolution tant du point de vue de ses domaines d'applications que dans ses technologies. Nous présenterons les principes de base de ces systèmes ainsi que les différents standards qui y sont liés. Nous verrons notamment le modèle de référence du Workflow Management Coalition (WfMC) ainsi que la spécification Workflow Management Facility (WMF) de l'Object Management Group (OMG). Nous verrons quels sont aujourd'hui les domaines d'applications et les caractéristiques qui ont permis aux systèmes de workflows d'élargir leurs champs d'applications. Nous conclurons ce chapitre par la présentation de quelques systèmes de workflows existants.

La deuxième partie, composée de trois chapitres, sera consacrée à la présentation de notre plate-forme. Nous commencerons dans le chapitre 4 par présenter l'architecture globale de notre système. Nous présenterons notamment son métamodèle et les différents concepts et technologies qui nous permettront de réaliser une plate-forme flexible. Nous terminerons par les différents constituants du moteur.

Le chapitre 5 se portera plus particulièrement sur les particularités de notre proposition. Nous nous intéresserons plus particulièrement à l'adaptabilité des modèles et de leurs instances en cours d'exécution. Nous verrons également les différentes opérations relatives à la gestion du temps et nous terminerons par la présentation d'une architecture multi-canal d'accès à notre système.

Le chapitre 7 concluera ce mémoire en résumant nos travaux et en montrant les différentes perspectives qui s'offrent à nous.

Première partie

Formation Ouverte et à Distance et Systèmes de Workflows

Chapitre 2

Langages de modélisation pédagogique

> The connection between the language in which we think/program and the problems and solutions we can imagine is very close. For this reason restricting language features with the intent of eliminating programmer error is at best dangerous.
>
> **Bjarne Stroustrup**

Les recherches récentes [Kop00, Eva01, WSKF02] montrent que les plates-formes de FOAD ne doivent pas se contenter d'apporter du contenu pédagogique car l'apprentissage ne se résume pas à un simple transfert de connaissances. La simple mise à disposition de ressources ne suffit pas. En effet, l'acquisition des connaissances provient de nombreuses sources et d'activités comme par exemple la résolution de problèmes, l'interaction avec de véritables instruments ou lors d'interactions entre les différents apprenants. Il est donc nécessaire de penser de manière globale au processus d'apprentissage car les activités des apprenants à l'intérieur d'un environnement d'apprentissage sont primordiales. Des langages de modélisation pédagogique ou EML (*Educational Modelling Language*) sont apparus afin de mieux gérer les ressources et de manière plus globale les activités pédagogiques. Ils permettent d'augmenter le niveau d'abstraction en se focalisant sur l'utilisation des différentes ressources plutôt que sur les ressources elles-mêmes. Il existe une grande diversité d'EML. Certains se focalisent plus sur la structuration des ressources pédagogiques dans un contexte donné alors que d'autres spécifient le cheminement complet de l'apprenant que cela soit à l'intérieur d'un module de formation ou entre les modules de formation. La caractéristique commune à tous ces langages est d'être le plus indépendant possible des plates-formes existantes afin que les modèles puissent être échangés et réutilisés avec un minimum de contraintes.

L'utilisation de ces langages représente la première étape du cycle de vie de la conception d'un enseignement à distance, car ils vont permettre de formaliser l'utilisation du contenu et ainsi autoriser sa mise en œuvre dans les plates-formes informatiques. Ils seront principalement utilisés par l'ingénieur pédagogique et parfois par le fournisseur de ressources pour présenter une offre structurée.

Le reste de ce chapitre est organisé de la manière suivante. Dans un premier temps, nous présenterons dans la section 2.1 quelques éléments de définitions des langages de modélisation pédagogique ainsi que leurs principales caractéristiques. Nous avons classé ces différents langages en trois catégories. La première catégorie, présentée dans la section 2.2, que nous avons intitulée *langages d'évaluation*, est composée de langages permettant de décrire de manière abstraite des phases durant lesquelles l'apprenant devra résoudre des problèmes ou répondre à des questions. La deuxième catégorie de langages (section 2.3), intitulée *structuration des ressources*, est composée de langages permettant d'agencer les différents contenus et ainsi de les fournir aux apprenants au bon moment dans leurs parcours pédagogique. Nous terminerons par la troisième catégorie (section 2.4) intitulée *structuration des activités pédagogiques*. Elle se focalise sur l'activité d'apprentissage elle même et le travail à réaliser. La section 2.5 concluera ce chapitre en montrant un modèle de référence possible et en indiquant notamment quel langage nous choisirons pour la suite de nos expérimentations.

2.1 Présentation des EML

Dans les plates-formes dédiées à l'éducation, on peut considérer quatre types de données [Tee02b] :

- le *contenu* qui doit être appris ou utilisé pour l'apprentissage ;
- les *descriptions de contenu* qui permettent d'identifier le contenu et notamment de connaître son objet d'étude ;
- la *description des processus d'apprentissage* qui définit les stratégies utilisées et les interactions entre enseignants et apprenants ;
- les *données organisationnelles* qui décrivent les groupes d'apprentissage (profils des différents membres, état d'avancement du groupe et l'historique d'apprentissage).

L'origine du terme EML provient des travaux réalisés à l'Open University of the Netherland sur la conception et le développement d'un langage de description adapté à l'éducation et nommé EML. Pour différencier l'appelation générique du langage de l'Open University, nous nommerons ce dernier EML-OU. EML-OU ainsi que l'ensemble des EML sont définis de la manière suivante par le CEN/ISSS WS/LT Learning Technologies Workshop [RvRK+02] :

> *« An EML is a semantic information model and binding, describing the content and process within a 'unit of learning' from a pedagogical perspective in order to support reuse and interoperability »*

Cette définition est à l'origine celle de EML-OU que nous présenterons dans la section 2.4.1. Si nous la décomposons un peu, nous voyons apparaître les éléments fondamentaux suivants :

Semantic information model and binding. Le 'binding' correspond à l'expression du modèle de manière formelle, dans un langage compréhensible par un ordinateur. Cette formalisation est réalisée le plus généralement avec XML.

Unit of learning. Le concept de 'unit of learning' est l'élément central. Il définit le modèle d'apprentissage, les ressources et les services requis afin d'atteindre un ou plusieurs objectifs étroitement liés. Il ne peut pas être réduit à ses composants internes sans perte de sa sémantique et de son efficacité pour atteindre les objectifs pédagogiques. Il peut correspondre à un cours, à un module de formation ou à une leçon.

Pedagogical perspective. Un EML doit être relativement indépendant des diverses théories d'apprentissage et par corrélation, il doit être possible d'utiliser différentes théories avec un EML.

Reuse and interoperability. Les EMLs ne doivent pas contenir d'éléments contextuels spécifiques empêchant leurs réutilisations ultérieures. Ils doivent pouvoir être réutilisés de manière transparente entre différents systèmes.

2.1.1 Caractéristiques principales

Koper [Kop01] définit 11 caractéristiques principales que devrait posséder un langage de modélisation pédagogique :

1. Le système de notation doit décrire les unités d'apprentissage d'une manière formelle afin qu'un traitement automatique soit possible (*formalisation*).
2. Le système de notation doit être capable de décrire des unités d'apprentissage qui sont basées sur des théories et des modèles d'apprentissage et d'instruction différents (*flexibilité pédagogique*).
3. Le système de notation doit exprimer de manière explicite la sémantique des différents objets d'apprentissage à l'intérieur du contexte d'une unité d'apprentissage. Il doit fournir la structure sémantique du contenu ou de la fonctionnalité des objets d'apprentissages associés à une unité d'apprentissage (*Objets d'apprentissage explicitement typés*).
4. Le système de notation doit être capable de décrire complètement une unité d'apprentissage. Ceci inclut tous les types d'objets d'apprentissage, les relations entre ces objets et les activités ainsi que le flot de travail des apprenants et des membres de l'encadrement avec les objets d'apprentissage (*complétude*).
5. Le système de notation doit décrire les unités d'apprentissage de telle manière qu'une exécution répétée soit possible (*reproductibilité*).
6. Le système de notation doit être capable de décrire des aspects de personnalisation à l'intérieur des unités d'apprentissage. Ainsi le contenu et les activités présents dans une unité d'apprentissage peuvent être adaptés par rapport aux préférences, aux compétences et connaissances antérieures et aux besoins éducationels des utilisateurs. De plus, le contrôle de la personnalisation doit pouvoir être donné soit aux étudiants, soit aux membres de l'équipe pédagogique, soit aux concepteurs (*personnalisation*).

7. La notation des composants de contenu, si possible, doit être neutre vis-à-vis du médium. Ainsi, elle peut être utilisée dans différents formats de publication comme par exemple le format web, le format papier, sous forme de e-book, *etc.* Elle doit également supporter différents environnements comme la formation à distance, l'apprentissage en ligne, l'apprentissage mobile, *etc* (*indépendance du médium et du cadre d'utilisation*).
8. Lorsque cela est possible, il est nécessaire de conserver une indépendance entre les standards qui sont utilisés pour la notation des unités d'apprentissage et les techniques utilisées pour interpréter la notation des unités d'apprentissage. Grâce à cela, les investissements dans le développement éducationel deviendront résistants aux changements techniques et aux problèmes de conversion (*interopérabilité et maintenance*).
9. Le système de notation doit s'accorder avec les standards et les spécifications existants (*compatibilité*).
10. Le système de notation doit permettre si possible l'identification, l'isolement, la décontextualisation et l'échange des objets d'apprentissage et leurs réutilisation dans d'autres contextes (*réutilisabilité*).
11. Le système de notation doit permettre si possible de produire, de modifier, de préserver, de distribuer et d'archiver les unités d'apprentissage et tous les objets d'apprentissage associés (*gestion du cycle de vie*).

2.1.2 Caractéristiques pour l'exécution

Les caractéristiques que nous avons présentées dans la section précédente correspondent aux propriétés statiques des modèles. Dans le cadre de nos travaux, nous nous intéressons plus particulièrement aux aspects exécution de ces langages. Nous avons isolé deux caractéristiques qui nous semblent intéressantes au sein d'un EML :

Gestion des groupes. Il existe une quantité non négligeable d'activités pédagogiques réalisées en groupe. Il est donc nécessaire de les gérer. Afin d'offrir plus de liberté aux apprenants, il est également intéressant de pouvoir spécifier des ensembles d'activités individuelles.

Gestion du temps. Le temps est un élément important dans le cadre de la formation, aussi bien au niveau de la gestion administrative des formations qu'à l'intérieur des modèles. Par exemple, dans le cadre d'une formation de groupe, la gestion du temps évite en partie de trop importants écarts dans l'avancement des travaux de chaque étudiant. Il permet également de mieux planifier les différentes tâches à réaliser. Cette gestion temporelle s'applique plus particulièrement à la durée d'exécution des activités ou aux périodes pendant lesquelles l'accès aux ressources est autorisé.

Nous allons maintenant étudier des EMLs appartenants aux trois catégories que nous avons définies en allant du plus spécifique au plus généraliste. Nous ne cherchons pas à être exhaustif mais à montrer un panorama des langages existants.

2.2 Langages d'évaluation

Comme nous l'avons présenté dans la section précédente, un questionnaire ou une évaluation sont considérés comme des unités d'études. De ce fait, les langages de description de questionnaires peuvent être considérés comme des EMLs. Nous allons donc nous intéresser maintenant à deux de ces langages, en présentant TML et IMS QTI.

2.2.1 TML

Le *Tutorial Markup Language* (TML) [Bri, TMLb] a été développé au Royaume-Uni à l'université de Bristol. L'objectif principal de ce langage est d'offir un moyen d'expression des questionnaires indépendant du système de représentation, i.e, il existe une séparation entre le contenu et la présentation des informations.

2.2.1.1 Méta-modèle

TML est un surensemble de HTML [TMLa]. Il reprend l'ensemble des balises existantes en y ajoutant les différents éléments caractéristiques permettant d'exprimer un questionnaire à choix multiple. Nous avons représenté en UML, figure 2.1, les différents concepts de TML.

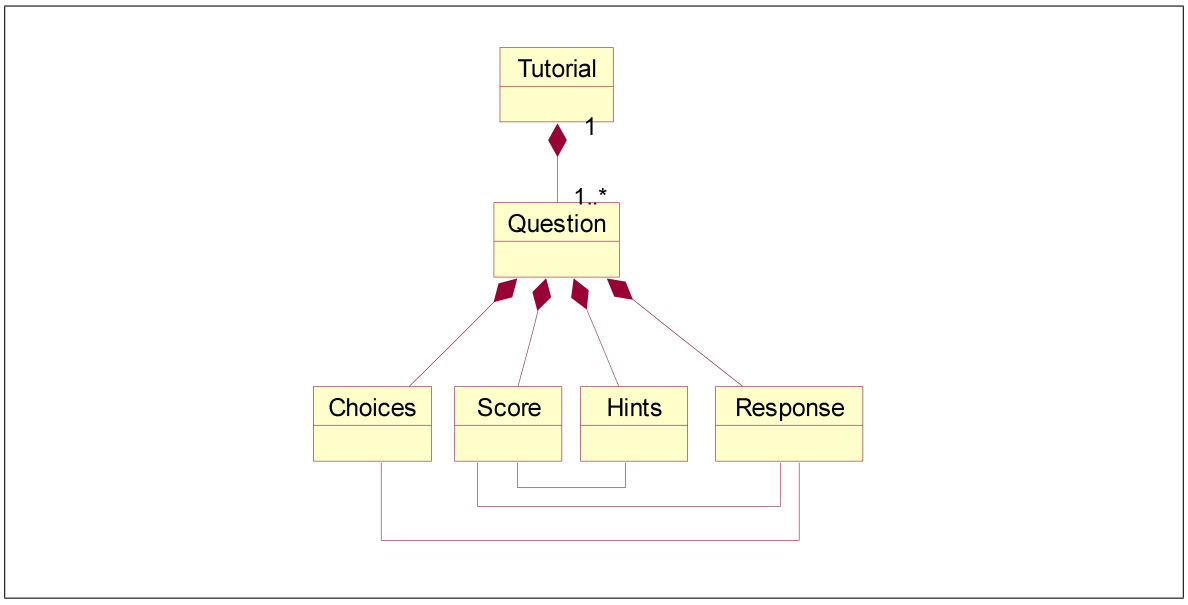


FIG. 2.1 – Méta-modèle de TML

L'élément de base est le *Tutorial*. Il contient l'ensemble des éléments *Question*. Chaque question est composée de quatre éléments, *Choices* qui représente les différentes réponses ; *Score* qui détermine les points obtenus pour une bonne et une mauvaise réponse ainsi que les points perdus lors de la demande d'un conseil ; *Hints* qui représente les différents conseils que peut demander un étudiant pour répondre ; et enfin *Responses* personnalisable pour chaque

réponse incluse dans *Choices*. Cette entité permet par exemple à l'enseignant d'expliquer pourquoi la réponse choisie est incorrecte.

2.2.1.2 Conclusion

Cet EML est très spécifique car il se limite aux questionnaires à choix multiples. Il ne contient pas de notion temporelle d'exécution ni d'ordonnancement précis des questions. Ce dernier point est à modérer car bien que le langage en lui même ne s'intéresse pas à l'ordre des questions, le système s'occupant de l'affichage des questions peut choisir de les présenter de manières différentes, toutes sur une seule page, ou une question à la fois. Dans ce cas l'ordonnancement est purement séquentiel car il n'est pas possible d'exprimer des conditions et des chemins alternatifs. Ce langage ne permet à aucun moment d'exprimer ou d'indiquer de quelles manières doivent être présentées les questions. De même, rien n'est prévu pour le traitement des réponses. Nous sommes donc confrontés au même problème qu'avec HTML dont le rendu dépend fortement du navigateur utilisé.

2.2.2 IMS Question & Test Interoperability

IMS Global Learning Consortium est une organisation dont l'objectif est de promouvoir le développement de l'apprentissage à distance à travers la coopération et la collaboration des universités, des agences gouvernementales et des sociétés qui travaillent pour l'évolution et l'utilisation d'environnements d'éducation à distance. IMS est concerné principalement par la définition et la maintenance de spécifications techniques qui répondent aux besoins des différents acteurs de la formation à distance. Les spécifications permettent à ces derniers de partager une base commune permettant notamment l'intéropérabilité et la réutilisabilité entres les différentes plates-formes. Les principaux domaines de normalisation sont l'échange des ressources pédagogiques, des profils des étudiants, des questionnaires ainsi que l'échange des conception des modules d'apprentissages.

L'objectif principal de *IMS Question & Test Interoperability* (IMS-QTI) [IMS02a] est de permettre l'échange de questions et d'évaluations entre différents LMS, en définissant différents schémas XML pour assurer l'intéropérabilité [IMS02d]. Cela permet de créer une banque de questions et de tests permettant la réutilisation et la composition ainsi que l'échange des résultats.

2.2.2.1 Méta-modèle

Le méta-modèle de IMS-QTI est présenté à la figure 2.2. Nous allons maintenant voir plus en détail ses différents constituants.

Item. C'est la plus petite unité indépendante. C'est une combinaison de questions, d'informations sur leur rendu (présentation des questions à l'utilisateur), le traitement des réponses et leurs notations et le retour d'informations (conseils ou solutions) présentées à l'utilisateur. Il est défini comme le bloc fondamental qui contient une ou plusieurs questions et réponses.

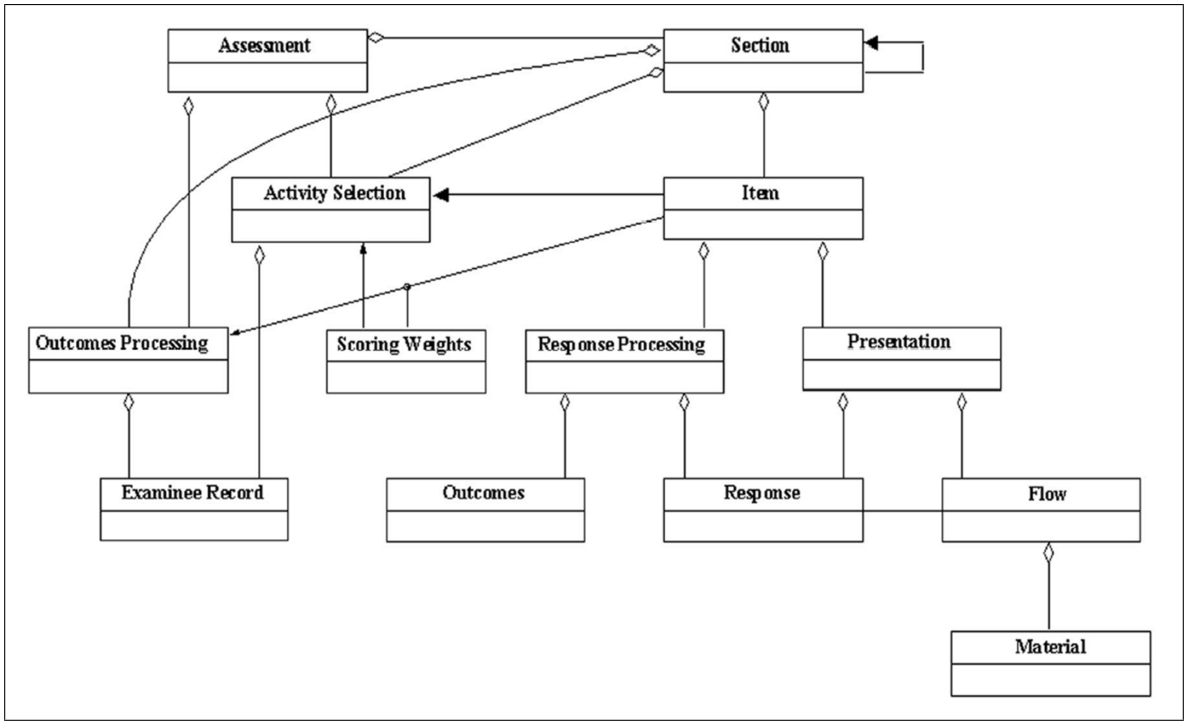


FIG. 2.2 – Méta-modèle de IMS QTI (tiré de [IMS02b]

Section. C'est une collection d'*Items* et de *Sections*. Elles permettent la construction de regroupement par exemple représentant un sujet particulier. Elles contraignent la longueur des instructions de séquencement et contrôlent la manière dont les différentes séquences sont construites.

Assessment. C'est une collection de *Sections*. C'est l'élément de plus haut niveau et il est unique pour un modèle particulier. Il ne peut pas avoir de lien avec un autre *Assessment* et il ne peut pas contenir directement un *Item*. Il contient obligatoirement au moins une *Section*. Il contient toutes les instructions nécessaires pour permettre le séquencement variable des *Items* et les scores correspondants afin de produire la note finale.

Activity Selection. Cette entité a pour but de déterminer le prochain élément à réaliser. Le choix s'effectue en fonction du chemin déjà parcouru et des résultats obtenus jusqu'au moment de la sélection.

Outcomes Processing. Les *Outcomes Processing* permettent de réunir et réconcilier tous les résultats d'évaluation obtenus afin de produire et présenter un résultat global de la section ou de l'assessment.

Examinee Record. L'*Examinee Record* reprend l'ensemble des résultats collationnés d'un étudiant. Il peut comprendre les résultats d'un seul processus complet d'évaluation ou être un enregistrement « tout au long de la vie » qui contient l'historique de la progression de l'individu.

Scoring Weights. Les *Scoring Weights* représentent les coefficients des résultats obtenus lors d'une évaluation.

Response Processing. Le *Response Processing* a pour objectif le traitement et l'évaluation des différentes réponses fournies par l'apprenant.

Outcomes. Les *Outcomes* représentent l'ensemble des résultats qui doivent être évalués par l'objet *Reponse Processing*. Cela détermine la métrique de notation à appliquer pour l'évaluation des réponses.

Response. Les réponses qui sont fournies par l'utilisateur aux différentes questions (*Item*), i.e., les sélections d'entrée de l'utilisateur.

Presentation. Cet élément contient toutes les instructions pour la présentation des questions et des réponses possibles durant une évaluation. Ces informations comprennent les différents contenus à présenter.

Material. Les *Materials* sont les différentes ressources qui seront utilisées par le moteur de question. Les différents types de contenus supportés sont les textes, les images, les vidéos, les documents audio, les applications externes et les applets java.

Flow. Un *flow* est défini comme un ensemble de contenus qui doit être manipulé par le moteur de rendu comme un bloc logique. Un *flow* peut contenir d'autres *flows* et ainsi une hiérarchie complexe peut être construite. Ce découpage permet lorsqu'un *Item* contient de nombreux *Materials* d'organiser et de structurer finement leurs ordre de présentation à l'utilisateur. En plus des ressources, un *flow* contient les différents labels pour les réponses possibles qui seront utilisées par l'entité de traitement des réponses.

Les éléments *Item*, *Section* et *Assessment* disposent d'un attribut *Duration* permettant d'indiquer la durée maximale autorisée pour la réalisation de l'élément.

D'un point de vue technique cette spécification comporte deux composants, un composant pour décrire et gérer les objets d'évaluation (item, section, assessment) présentés au participant et un composant rendant compte des résultats de manière globale ou question par question.

La spécification *Selection & Ordering* [IMS02c] offre une possibilité supplémentaire d'exprimer la sélection et l'ordonnancemnt des sections et des items. Le processus de *Selection & Ordering* se découpe en trois phases. La première, nommée *object sequence*, consiste lors de la construction du modèle à déterminer les différentes contraintes à appliquer aux règles de sélection. Les deux phases suivantes qui se déroulent lors de l'exécution sont nommées respectivement *object selection* et *object ordering*. L'*object selection* consiste à sélectionner les objets pouvant être activés à un instant donné du processus d'évaluation. L'*object ordering* s'occupe de l'ordonnancement des différentes activités sélectionnées précédemment.

2.2.2.2 Conclusion

Cette spécification présente de nombreux avantages sur TML pour la description des composants d'évaluation. Les différences majeures avec TML résident dans les aspects présentation et structuration des données (avec en plus une notion temporelle pour chacun des composants Assessment, Section et Item). IMS QTI ne s'intéresse pas uniquement à la description des questions mais également à leur présentation tout en autorisant plusieurs représentations

pour une même question afin de conserver une certaine indépendance entre contenu et contenant. Le séquencement des questions [IMS02c] permet la spécification de l'ordonnancement des questions de très nombreuses façons. Il faut également noter que la spécification [IMS02b] contient des éléments de coordination (precondition, postcondition) dont la sémantique n'est pas encore complètement définie. Elle le sera dans la version 2.0 de la spécification.

IMS-QTI ne permet de modéliser qu'un seul type d'activité qui est l'activité d'évaluation. Cette spécification ne correspond donc pas exactement à nos attentes car nous souhaitons modéliser les processus d'enseignement dans leur globalité et non certaines activités particulières. Comme nous le verrons par la suite, nous pourrons utiliser cette spécification pour la définition précise d'activités.

2.3 Structuration des ressources

Nous allons maintenant présenter dans cette section différents langages permettant de structurer les différentes ressources pédagogiques qui seront manipulées par les apprenants. Nous commencerons par *Targeteam* qui, en plus de la structuration, est utilisé pour la conception des ressources. Puis nous verrons la proposition de l'AICC et nous terminerons par LMML auquel il est possible d'adjoindre des modèles pédagogiques particuliers afin d'élever le niveau d'abstraction.

2.3.1 Targeteam

Targeteam (TArget Reuse and GEneration of TEAching Materials) [Tar03] a pour objectif le support du travail avec des ressources pédagogiques. Il est destiné à gérer de larges collections de faits et d'autres contenus qui doivent être étudiés par les apprenants. Ces matériels incluent des annotations pédagogiques (explications, motivations, exemples) et ils sont structurés avec soin et étroitement liés. Targeteam est basé sur l'hypothèse que la structure est cruciale pour fournir une bonne compréhension des faits à l'apprenant. Un simple séquencement linéaire de modules isolés n'est pas suffisant. Targeteam a été conçu pour couvrir le spectre complet du matériel d'apprentissage, allant des fragments non structurés aux collections de contenus très fortement structurées. Il se focalise plus particulièrement sur la réutilisation, i.e., l'utilisation d'un contenu particulier aussi bien pour un cours d'introduction que dans le cadre de travaux pratiques. Même avec l'utilisation de méta-données, la réutilisabilité fonctionne mieux avec des éléments de faible granularité. Pour l'étendre, Targeteam ajoute les concepts d'abstraction et d'adaptabilité.

Afin d'être réutilisé, le contenu doit être indépendant du contexte d'exécution. À ce titre, Targeteam fournit différents mécanismes d'abstraction :

- Le premier mécanisme est de définir le contenu indépendamment de la présentation. Nous retrouvons cette séparation dans les langages que nous avons présentés précédemment.

- Le deuxième moyen mis en œuvre consiste à fournir une structure hiérarchique abstraite. Abstraite signifie que l'on ne parle pas de chapitre, de section ou de paragraphe. Ces notions apparaissent lors de la transformation du contenu en vue de sa présentation.
- Le troisième mécanisme est l'abstraction de l'utilisation. La ressource peut aussi bien être utilisée dans le cadre d'un apprentissage basé sur les problèmes ou présentée par un enseignant. Le contenu abstrait peut être déplacé facilement entre différents contextes. Cependant, il nécessite souvent d'être adapté au contexte. Un même contenu ne peut être utilisé de la même manière s'il est placé dans un cours d'introduction ou dans un exercice pratique.

L'adaptation ne se situe pas au niveau de la présentation mais au niveau du contenu. Le contenu peut être adapté en enlevant un morceau, en ajoutant un morceau pour repositionner le contenu dans le nouveau contexte ou en redéfinissant certains mots par des termes plus appropriés. Dans Targeteam cela signifie qu'une transformation est appliquée au contenu lorsqu'il est utilisé dans un contexte particulier. Cela permet de garder le contenu original intact et ainsi de profiter des mises à jour.

2.3.1.1 Méta-modèle

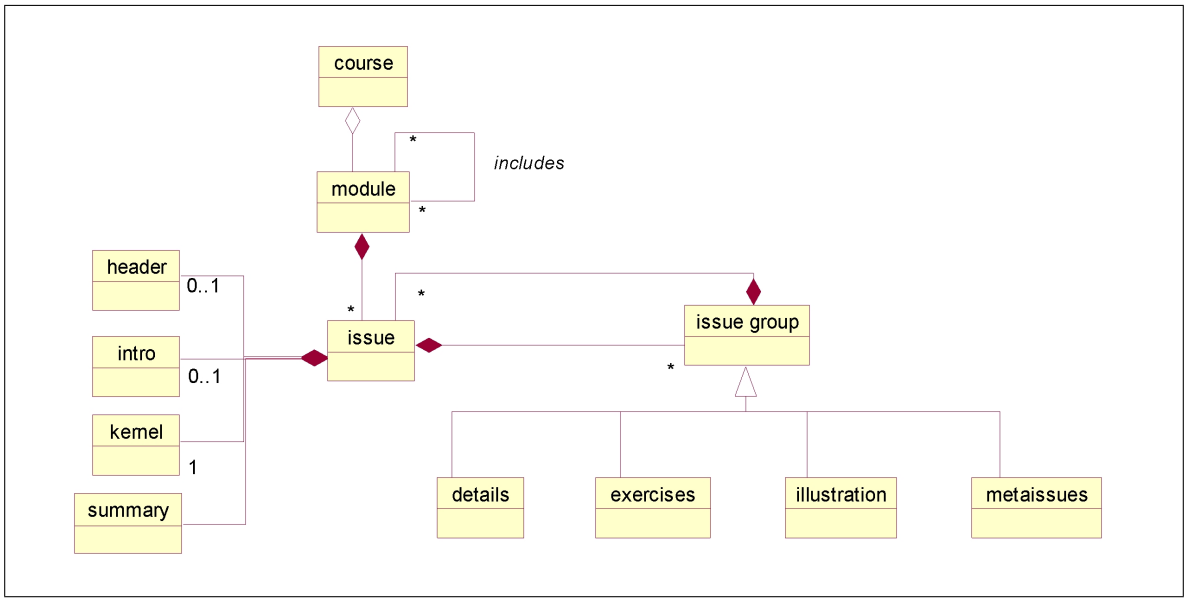


FIG. 2.3 – Méta-modèle de TeachML (tiré de [Tee02b])

Targeteam s'appuie sur un langage délibérément simple, TeachML [Tee02b, Tee02a], dont nous allons maintenant voir plus en détail le méta-modèle présenté à la figure 2.3. Un cours est composé d'unités réutilisables qui sont les *modules*. Un module peut lui-même faire référence à d'autres modules. Le contenu d'un module est organisé de manière thématique par l'intermédiaire d'*issue*. Cet *issue* peut être aussi grand qu'un livre entier ou aussi petit qu'une

simple phrase ou que quelques mots-clés. Il se compose d'un *header*, d'une *introduction*, d'un *kernel*, d'un *summary* et d'un certain nombre de *issue group*. Le *kernel* contient la spécification principale de l'*issue* sous forme d'un contenu non structuré. Il contient directement le texte ou les autres médias (images, son, ...). Le *header* correspond à une donnée textuelle, de taille inférieure au contenu du *kernel*. Cela correspond par exemple au titre de l'*issue*. L'*intro* contient un texte court fournissant les relations avec le contexte d'apprentissage ou la motivation de l'*issue*. *summary* contient un texte court résumant le contenu de l'*issue*. Les sous-groupes sont classés selon leur rôle dans l'*issue*. La structuration d'un *issue* est réalisée par les *issue group*.

Les *details* contiennent une séquence de *issue* qui fournissent des aspects plus spécifiques de l'*issue* contenante. Ils sont les parties thématiques de l'*issue* bien qu'elles ne soient pas formulées explicitement dans le *kernel*. L'*illustration* contient une séquence de *issue* qui illustre l'*issue* les contenant. Une *illustration* peut être un exemple, une figure ou un *issue* complexe qui décrit une application de l'*issue* contenant. Les *exercises* contiennent une séquence de *issue* qui peut être utilisée par l'apprenant pour s'entraîner. Un exercice peut être une simple question ou il peut être un problème complexe à résoudre. Les *metaissues* contiennent une séquence de *issue* avec des informations sur l'*issue* contenante. Cela peut être des informations sur la manière de l'utiliser, la raison de sa présence, son origine, *etc.*

2.3.1.2 Conclusion

Targeteam offre des avantages pour la gestion des contenus et leurs utilisations dans différents contextes d'apprentissage et indépendamment du type d'activité, contrairement aux deux langages vus précédemment. Grâce aux différents mécanismes de transformation utilisés avec ce langage, il permet une bonne réutilisabilité et une grande adaption aux différents contextes.

Bien que faisant partie des langages présentés dans [RvRK+02], nous le trouvons à la limite des EMLs car l'aspect réellement lié à l'éducation est très restreint. Les différences avec des langages de mise en page de contenu, comme par exemple LaTeX, sont très faibles. De plus la structuration est sous la forme d'un arbre à suivre, sans possibilité d'exprimer directement des conditions de passage d'un contenu à l'autre. Il est impossible de réaliser un véritable parcours entre les différents éléments d'apprentissage.

2.3.2 AICC CMI Data Model

L'*Aviation Industry CBT*[2] *Commitee* (AICC) [Avi03] est une organisation internationale, fondée en 1988, regroupant tous les intervenants dans le domaine de la formation de l'industrie aéronautique (constructeurs d'avions, vendeurs de plates-formes informatiques d'enseignement, fournisseurs de contenu pédagogique, *etc*). Cette association est cependant ouverte aux personnes provenant d'autres industries. Les standards proposés sont par nature génériques et ne se focalisent pas uniquement sur le domaine aéronautique. L'AICC publie des *AICC*

[2] **C**omputer **B**ased **T**raining

Guidelines & Recommendations (AGR) dans différents domaines comme par exemple l'AGR-007 [AIC95] qui traite de l'échange de contenus entre plates-formes ou l'AGR-002 [AIC02] qui traite de la plate-forme matérielle d'un poste client (vitesse du microprocesseur, espace disque, *etc*). Il existe à l'heure actuelle 9 AGRs différentes. Celle qui nous intéresse le plus est l'AGR-006 [AIC98] qui traite de l'interopérabilité entre *Computer Managed Instruction* (CMI). AICC définit un CMI comme un système gérant à la fois le comportement connecté (on-line) et déconnecté (off-line) des activités d'apprentissage. Il peut notamment s'appuyer sur des CBTs. L'AGR-006 décrit notamment l'échange de fichiers représentant la structure d'un cours. Elle repose principalement sur le document [AIC01].

2.3.2.1 Constituants

Afin de mieux comprendre le fonctionnement et l'architecture d'un CMI nous allons présenter ses cinq principaux constituants, puis nous nous intéresserons aux formats d'échange de la structure d'un cours. Un CMI est constitué de :

Un composant de développement de la structure des cours. Ce composant se situe au centre des systèmes CMI. Il contient des outils permettant la définition d'une liste de leçons et leurs attributs comme par exemple le type de leçon (connecté/déconnecté), les objectifs pédagogiques ou les ressources qui seront utilisées. La structure des cours fournit une méthode pour grouper des leçons en séquences pour l'assignation. Ce support d'une hiérarchie autorise la définition de relations de précédence ou de successeur. Il est important de pouvoir définir des hiérarchies complexes. Pour cela, il est intéressant de grouper les leçons dans un élément de plus haut niveau, appelé *bloc*, et de pouvoir constituer la hiérarchie du cours en s'appuyant sur les leçons et les *blocs*. Le CMI peut être capable de s'adapter pour permettre l'ordonnancement et l'assignation des leçons par différents CBT.

Un composant d'évaluation. L'évaluation est souvent une partie d'une leçon et est généralement traitée comme une fonction externe du CBT. Elle peut être réalisée en mode connecté, avec l'aide des outils du CBT, ou déconnecté en remplissant par exemple un questionnaire papier dont la correction pourrait être assurée par un lecteur optique. Une évaluation est composée d'éléments d'évaluation ciblant des objectifs particuliers. Cette décomposition autorise un fonctionnement souple de ce composant en permettant par exemple de définir le nombre d'éléments corrects nécessaires pour valider l'évaluation, ou des questions critiques entraînant automatiquement l'échec de l'évaluation en cas d'erreur. Elle permet également de faire varier les poids associés à chaque élément. Les évaluations peuvent avoir lieu avant un cours afin de déterminer si l'étudiant peut être dispensé de certaines leçons, ou elles peuvent être effectuées après les leçons pour vérifier les connaissances acquises par l'étudiant.

Un composant de gestion des étudiants. Ce composant supporte la déclaration et l'inscription des étudiants. Il permet également de connaître les cours dans lesquels un étudiant est inscrit. Il peut également fournir des mécanismes d'inscription de masse pour par exemple inscrire une classe complète ou des mécanismes d'auto-inscription permettant à un étudiant de s'inscrire sans intervention de l'enseignant. Il supporte la

désinscription des étudiants avec suppression ou non des données qui leurs sont relatives (données gardées à des fin d'analyse a posteriori). Ce composant peut être utilisé à des fins administratives pour obtenir la liste des étudiants inscrits à un cours ainsi que pour connaître les leçons actuellement assignées aux étudiants. Il permet d'analyser les données récupérées lors de la réalisation d'un cours et il permet de dresser une carte du cours (cheminement utilisé, ...).

Un composant de gestion de l'assignation des étudiants. Ce composant permet de réaliser un suivi journalier des travaux des apprenants. Il autorise l'assignation par le système ou par un enseignant des différentes *Assignable Units* (AUs) (les leçons), qui sont les plus petits éléments assignables par un CMI. Il fournit également aux étudiants l'accès au système. Ils comprend trois grandes catégories de fonctions :

- **Les fonctions pour l'enseignant et l'administrateur** se composent principalement de méthodes autorisant un enseignant à valider ou à modifier le parcours prévu pour un étudiant. Par exemple, lorsqu'un étudiant a réalisé le maximum d'essais autorisé pour une leçon sans toutefois l'avoir réussie, l'enseignant peut afin de ne pas laisser l'étudiant bloqué réaliser une validation manuelle (avec un statut à *fail*) de la leçon et réorienter ce dernier vers d'autres leçons adaptées aux difficultés de l'apprenant.
- **Les fonctions d'assignation du système** se constituent principalement de l'ordonnanceur. Il enregistre la progression de l'étudiant, détermine les leçons suivantes et réalise l'assignation. Il dispose également d'un composant de gestion et d'allocation des ressources qui sont utilisées par les différentes leçons.
- **Les fonctions de connexion de l'étudiant** offrent à l'étudiant un point d'accès unique au système. Leur objectif principal est la présentation à l'étudiant des différentes leçons qui lui sont assignées et de leurs manipulation (démarrage, *etc*).

Un composant de gestion et collecte des données. Ce composant assure la récolte des informations et leur gestion. Il manipule différents types de données comme les informations relatives au déroulement des leçons ou les résultats des évaluations des étudiants.

Après cette présentation des CMI, nous allons maintenant nous intéresser à un des aspects de l'interopérabilité entre ces systèmes : l'échange des cours. Un cours se définit aussi bien comme une simple succession séquentielle de leçons que comme une hiérarchie complexe de centaines de leçons dont certaines comportent des pré-requis (exécution d'autres leçons précédemment) et d'autres peuvent être réalisées librement à n'importe quel moment. Les trois constituant basiques d'un cours sont :

- Les **éléments d'instruction** également appelés contenu. Cela comprend notamment les leçons, les tests et toutes les autres unités assignables. Souvent une description du contenu contient la liste des objectifs à atteindre du cours.
- La **structure** qui détermine l'ordre dans lequel les unités assignables doivent être réalisées par l'étudiant. L'ordre peut être simple (exécution séquentielle) ou complexe (chemin conditionnel dépendant par exemple du résultat de l'évaluation, notion de pré-requis entre les leçons).

– Le **comportement** correspond à la logique de progression à l'intérieur d'une leçon et également entre deux leçons.

Ces éléments sont indissociables car ils représentent un tout formant le cours. Si l'un des éléments venait à disparaître, cela remettrait en cause l'intégrité du cours et ainsi son sens et pourrait provoquer une perte de son efficacité.

2.3.2.2 Méta-modèle

Afin de faciliter l'échange des différents modules de cours, l'AICC a défini 7 fichiers d'échanges au format ASCII. Cela s'explique par la relative ancienneté de la spécification dont la première version date de 1993, avant l'apparition de XML. Chacun des fichiers prend en charge une partie du modèle. Afin de mieux comprendre les différents concepts manipulés par ces fichiers, nous avons réalisé un diagramme UML reprenant l'ensemble des entités que nous allons maintenant détailler (figure 2.4).

Course. Cette entité contient les différentes informations relatives à un cours. On y trouvera des informations administratives comme le nom de l'auteur, l'identifiant du cours ou son numéro de version. On rencontrera également des informations sur le cours lui même comme son titre, sa description (contenant son champ d'application, un résumé des principaux objectifs, ...) et le nombre d'unités assignables qu'il contient.

Descriptor. Cette entité contient la liste de tous les éléments de cours (Assignable Units, blocs, objectif) d'un cours particulier. Il contient pour chaque élément sa référence dans le cours, sa référence dans le référentiel ainsi que son titre et une courte description.

Assignable Unit. Cette entité est définie comme l'élément atomique représentant par exemple une évaluation ou un cours. Elle contient essentiellement une référence sur une ressource pédagogique, plus précisément le nom d'un exécutable DOS avec ses options. Elle est caractérisée par un type. Ces types sont définis par le concepteur du modèle. Il n'existe pas de référentiel de termes à utiliser pour classifier les AUs. Chaque AU contient une notion temporelle. Il est possible de spécifier la durée maximale qu'un étudiant doit passer pour terminer le travail qui lui est assigné. Lorsque ce temps est atteint, deux actions sont possibles, quitter l'AU ou continuer à travailler dessus. Ces actions peuvent ou non avertir l'étudiant par un message.

Block. Cette entité permet de structurer les différents éléments d'un cours. Un bloc est un regroupement d'AUs et de blocs. Ce regroupement n'implique pas d'ordonnancement, i.e., il n'existe pas de séquencement explicite à l'intérieur d'un bloc, même si souvent il correspond à l'exécution séquentielle des différents membres d'un bloc. L'ordonnancement sera réalisé à l'aide des prérequis (*Prerequisites*), des objectifs (*Objectives*) et des caractéristiques de terminaison (*Completion Requirements*). Cela met bien en évidence la différence entre la structuration et le comportement. Un bloc structure alors que les prérequis, les objectifs et les caractéristiques de terminaison créent le comportement.

Objective. Cette entité est principalement composée d'un titre est d'une description fixant les buts à atteindre. Une leçon peut couvrir plusieurs objectifs. Un objectif peut nécessiter la réalisation de plusieurs leçons. Un objectif peut être composé de sous-objectifs.

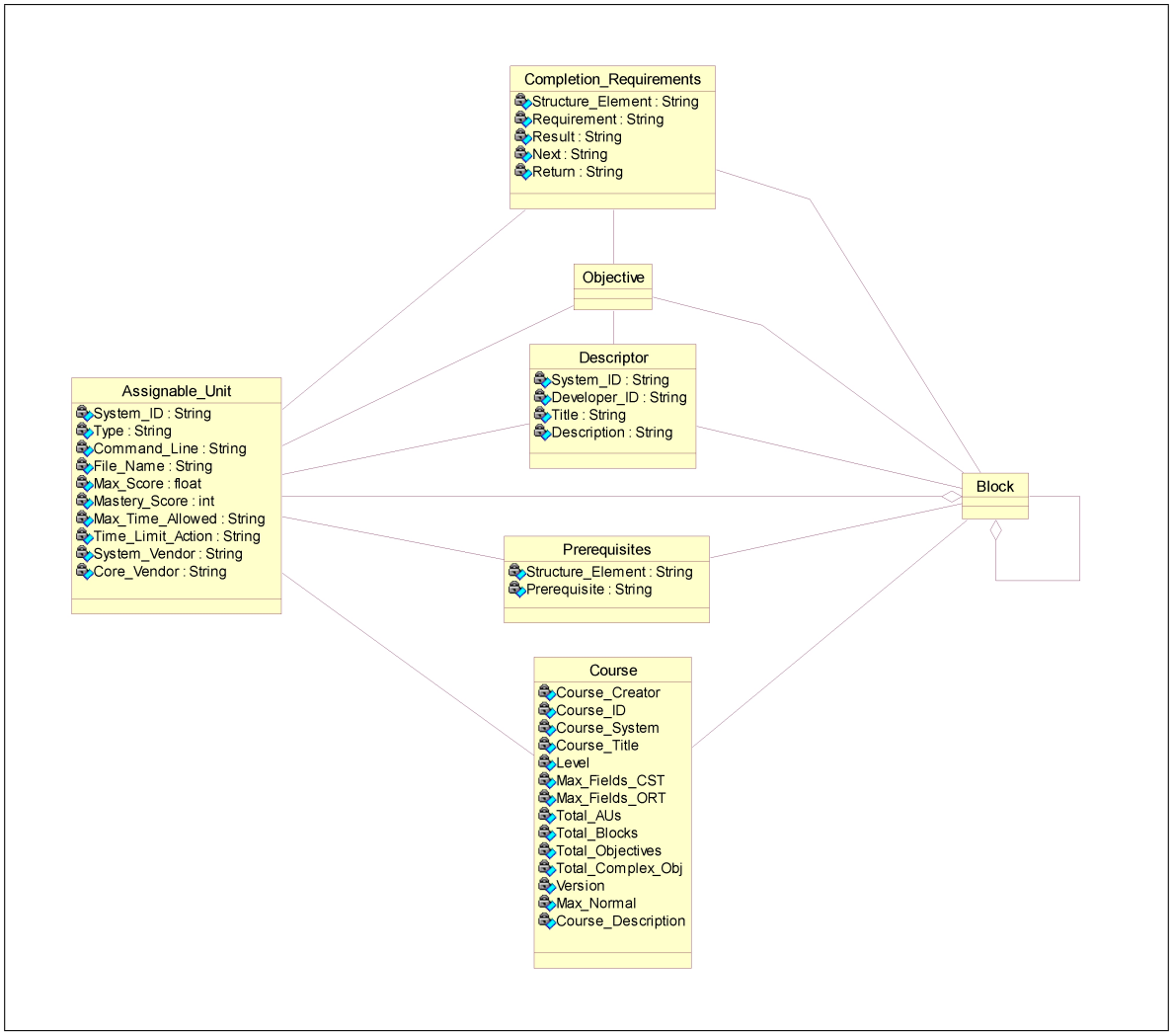


FIG. 2.4 – Méta-modèle pour l'échange de modèles de cours entre CMI

Prerequisites. Les prérequis peuvent être définis en terme de leçon terminée ou en terme d'objectif atteint. Parfois il peut être désirable d'empêcher la réalisation d'une leçon par un étudiant avant qu'il n'ait rempli certains prérequis. Ces derniers permettent d'exprimer des contraintes et ainsi l'ordonnancement entre les différents AU et blocs. Chaque contrainte est une expression qui identifie les éléments de cours qui déterminent si l'étudiant peut commencer le bloc ou l'AU. Les prérequis sont une liste des éléments de cours qu'un étudiant doit avoir *réussi* ou *terminé*. *Réussi* et *Terminé* sont des états. L'état d'une leçon est souvent déterminé par sa logique interne. Pour chaque leçon il y a six états possibles (*passed*, *completed*, *browsed*, *failed*, *not attempted*, *incomplete*). Chaque expression logique peut se référer aux états précédents. Le fonctionnement est identique avec des objectifs. L'état des objectifs permet également de déterminer si un étudiant rempli les conditions de prérequis. Une expression logique est une liste

d'éléments de cours (blocs, AUs, objectifs) avec leurs états séparés par des opérateurs logiques (et, ou, non, n parmi m).

Completion Requirements. Bien que l'état d'une leçon ou d'un objectif est souvent déterminé par sa logique interne, ce n'est pas toujours le cas. Par exemple, il peut y avoir des AUs conçues pour pré-tester l'étudiant. En démontrant sa capacité dans ces pré-tests, l'étudiant peut valider implicitement des parties de leçons ou même les obtenir complètement sans jamais les avoirs vues. En d'autres mots, le CMI peut parfois déterminer l'état d'un élément par des facteurs externes à l'élément lui-même. De la même façon, l'état d'un bloc ou d'un objectif complexe est défini par d'autres éléments de la structure. Ansi, l'état d'un bloc ou d'un objectif complexe doit être déterminé par le CMI. Les *caractéristiques de terminaison* sont conçues pour permettre la spécification explicite du moment où une AU, un bloc ou un objectif doit être assigné. Cette entité comprend un attribut *requirement* contenant une expression logique permettant de déterminer l'état de l'AU/bloc. Elle reprend la même syntaxe que pour l'expression des prérequis. L'attribut *result* indique l'état de l'élément lorsque l'expression logique est évalué à `TRUE`. L'attribut *next* identifie la leçon qui doit être lancée automatiquement lorsque l'évaluation de l'expression est `TRUE`. L'attribut *return* indique quelle AU le CMI doit retourner après la complétion de l'AU spécifiée dans l'attribut *next*. Cela permet par exemple en cas d'échec de lancer une nouvelle activité puis de revenir dans l'activité qui a posé problème. Cette entité permet l'ordonnancement explicite des différents éléments.

2.3.2.3 Conclusion

Nous venons d'exposer la proposition de l'AICC pour la modélisation et l'échange de cours dont l'objectif principal est l'interopérabilité entre systèmes. Cette proposition offre une modélisation des cours plus générique que les langages précédents. Elle est à mi-chemin entre la simple structuration de ressources et la description et l'ordonnancement d'activités pédagogiques. Elle pose les principaux concepts que nous retrouverons dans différents langages présentés ultérieurement, comme par exemple les notions d'objectif et de prérequis. Celles-ci permettent ainsi de mieux guider l'étudiant qui connaît les buts à atteindre et l'ordre de réalisation des différentes activités. Cette proposition présente également des notions temporelles en permettant de spécifier des limites et des durées pour les AUs.

Ce qui pourrait être l'un des inconvénients majeurs pour un pédagogue est l'impossibilité d'attacher le modèle de cours à un modèle pédagogique particulier. D'un point de vue technologique il présente l'inconvénient de ne pas proposer un mapping vers un langage basé sur XML contrairement à tous les autres langages présentés dans ce chapitre, cela s'explique par l'ancienneté de cette proposition.

2.3.3 LMML

Le *Learning Material Markup Language* [LMM03] a été développé en Allemagne à l'université de Passau. Il est basé sur le *Passau Teachware Model* qui est un méta-modèle décrivant la

structure modulaire générale de contenu de e-learning indépendant du modèle pédagogique. Il peut être instancié pour différents domaines d'applications comme les mathémathiques, l'informatique, la finance, *etc.*

2.3.3.1 Méta-modèle

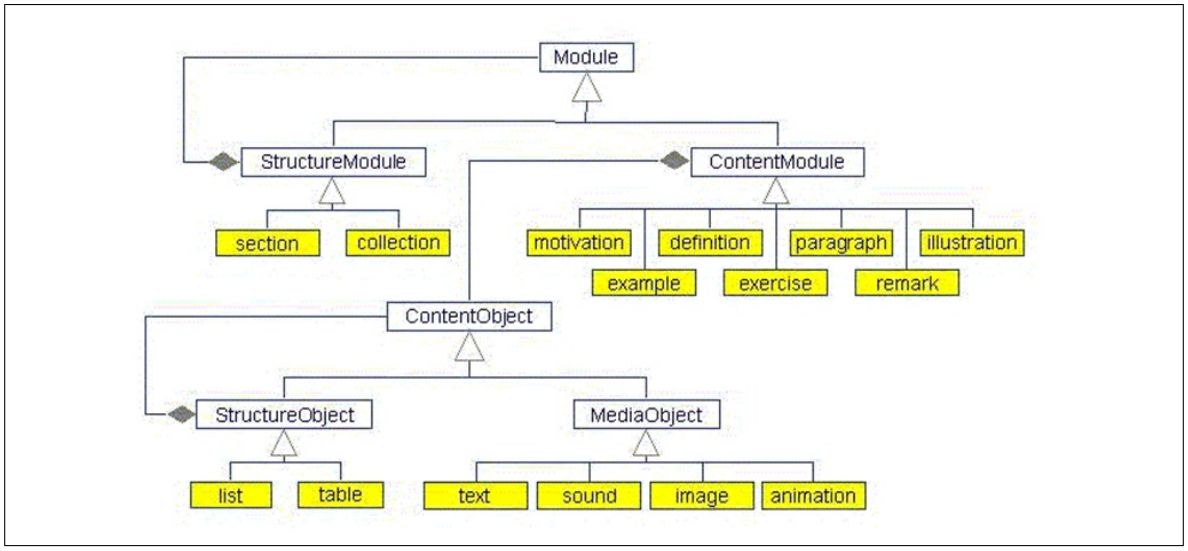


FIG. 2.5 – Méta-modèle de LMML (tiré de [LMM03])

Le contenu est organisé de manière modulaire [SF02, SFB99]. Il forme une hiérarchie consistant en *Modules* qui peuvent contenir d'autres modules. La plus petite entité de la hiérarchie pertinente pour un fournisseur de contenu pédagogique sont les *ContentModules* qui peuvent être par exemple des paragraphes, des définitions, des exercices, *etc.* Ils contiennent des *ContentObjects* qui peuvent être structurés en *liste* ou en *tableau.* Ils contiennent des *MediaObjects* qui peuvent être statiques ou dynamiques comme par exemple des images, des vidéos, *etc.*

Ce méta-modèle forme la base conceptuelle pour l'adaptation du framework LMML lui-même (instantiation et extension). Il facilite la réutilisation des différents composants et le développement coopératif de contenu pédagogique et supporte leur adaptation à différents contextes et aux besoins de l'étudiant. De plus il guide la création d'index pour la récupération de contenu stocké dans un référentiel XML. Enfin, certaines propriétés d'exécution d'un système de e-learning peuvent être spécifiées à un méta-niveau fournissant ainsi une forme de réflexivité qui autorise la modification du comportement du système, e.g., de forme ou de présentation du contenu.

La structure de LMML est un binding XML du méta-modèle présenté précédemment. Ces extensions spécifiques à un domaine sont des réalisations XML du modèle du domaine correspondant. Ainsi, LMML n'est pas uniquement un langage à balises mais plutôt une famille extensible et basée sur un modèle de langage XML pour le domaine du contenu pédagogique.

Dans LMML, le niveau du contenu, le niveau de navigation, le niveau de didactique et le niveau d'affichage et visualisation sont strictement séparés. Il est donc possible d'adapter par exemple l'affichage en fonction du contexte. Le même mécanisme supporte la publication multi-canal du contenu pédagogique. Il est possible d'étendre le méta-modèle en généralisant les entités existantes.

LMML se focalise sur la structuration conceptuelle et modulaire des ressources pédagogiques. Afin de faciliter un apprentissage efficace, il est nécessaire d'introduire un niveau d'abstraction supplémentaire qui modélise et décrit la structure didactique des ressources. LMML peut être combiné avec différents modèles pédagogiques. Le modèle didactique de référence utilisé par LMML est présenté à la figure 2.6. LMML peut être combiné avec différents modèles pédagogiques.

2.3.3.2 Méta-modèle didactique

Afin d'élargir le spectre d'utilisation de LMML et de le recentrer sur les aspects pédagogiques, il est possible de lui adjoindre un modèle pédagogique. Il permet de structurer les différentes activités d'apprentissage.

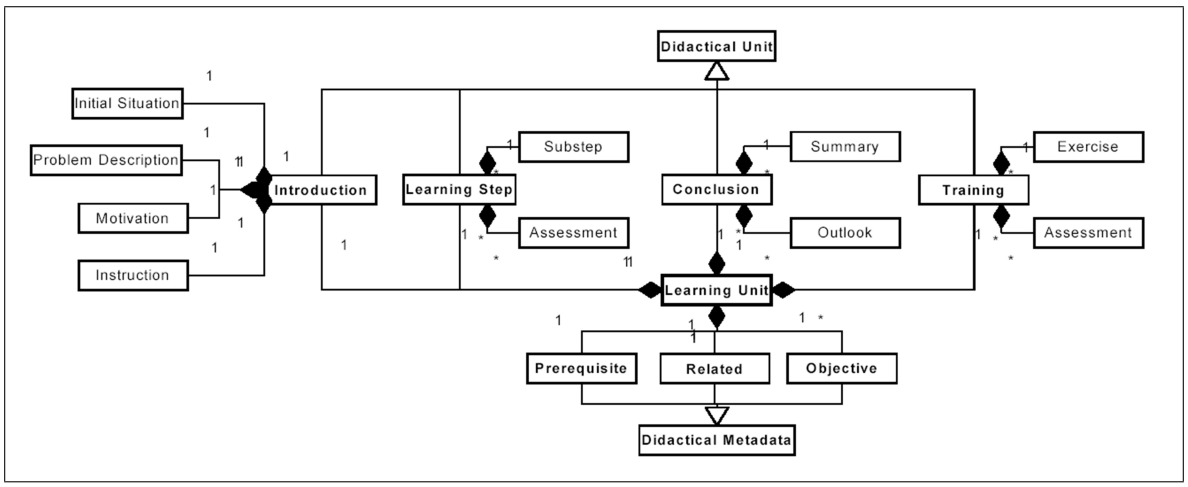


FIG. 2.6 – Méta-modèle d'une unité d'apprentissage (tiré de [WSKF02])

La décomposition est réalisée en deux niveaux. Un premier niveau appelé *module d'apprentissage* (*learning module*) représente un enseignement complet sur un thème précis. Il correspond à environ 2 à 10 heures de travail. Il est facilement réutilisable car il peut fonctionner de manière autonome. Le deuxième niveau appelé *unité d'apprentissage* (*learning unit*, figure 2.6) permet de décomposer le module d'apprentissage en séquences d'environ 45 minutes. Une unité d'apprentissage se compose de deux concepts principaux, des méta-données didactiques (*didactical metadata*) et des unités didactiques (*didactical unit*). Les méta-données didactiques contiennent des informations sur l'unité d'apprentissage permettant à l'apprenant de trouver une unité appropriée, de démarrer et de naviguer parmi les ressources selon ses

objectifs personnels. Ces données permettent également de spécifier des relations avec d'autres unités d'apprentissage ainsi que le profil de l'apprenant. Elles comportent les trois catégories suivantes :

Prerequisite. Cette entité représente les conditions requises concernant le profil de l'apprenant souhaitant réaliser l'unité d'apprentissage (son niveau, ses connaissances, ...).

Objectives. Cette entité représente les cibles de l'unité d'apprentissage selon la taxonomie de Bloom [Blo56] (connaissance, compréhension, application, analyse, synthèse, évaluation).

Related. Cette entité fait référence aux différentes ressources relatives à l'unité d'apprentissage. Elles peuvent être internes ou externes au module, être en-ligne (web) ou hors-ligne (livre), être des logiciels ou du matériel de travaux pratiques.

Une unité didactique représente les instructions de l'unité d'apprentissage. Elle comporte les éléments suivants :

Introduction. Cette entité fournit à l'étudiant le contexte de l'unité d'apprentissage au sein du module. Elle définit le problème à résoudre de l'unité, elle motive l'apprenant et elle le guide en lui fournissant des instructions sur la manière de travaillller avec les différentes ressources.

Learning Step. Cette entité représente les détails des concepts qui doivent être appris. Elle correspond à une durée approximative de 10 à 15 minutes et se décompose en un petit nombre de *substep*. Après avoir réalisé toutes les sous-étapes, l'étudiant est amené à réaliser une auto-évaluation (*assessment*).

Training. Cette entité correspond à un auto test permettant à l'étudiant de vérifier sa compréhension de l'unité.

Conclusion. Cette entité resitue le travail effectué, résume les points abordés et motive l'apprenant à continuer d'étudier les autres unités d'apprentissage.

2.3.3.3 Conclusion

Le travail sur LMML a été plus loin que celui des deux langages précédents dans le sens de la pédagogie car il peut intégrer un modèle didactique pour orienter le cheminement de l'étudiant et ceci dans une approche cognitive.

LMML est à la limite des langages de structuration de contenu et des langages de structuration des activités pédagogiques car il peut être couplé à un modèle pédagogique. LMML seul peut sembler moins intéressant que AICC car il ne se focalise que sur la structuration des ressources. Cependant il est possible d'y adjoindre des modèles pédagogiques permettant ainsi d'augmenter le niveau d'abstraction et autoriser une réelle modélisation tenant compte des divers aspects pédagogiques.

2.4 Structuration des activités pédagogiques

Dans cette section nous allons plus particulièrement présenter les langages dédiés à la description des activités pédagogiques et à leur structuration tout en étant relativement indé-

pendants des différents documents qui seront manipulés. Nous commencerons par présenter de manière globale *EML-OU* puis nous verrons en détail sa version standardisée par IMS, le *Learning Design Specification*.

2.4.1 Educational Modelling Language de l'Open University

L'*Educational Modelling Language* de l'Open University of the Netherlands a donné son nom et sa définition aux EMLs. L'origine de ces travaux provient de l'inexistence de moyen pour exprimer une formation en prenant en compte l'aspect pédagogique. La démarche est large car elle ne se concentre pas uniquement sur les systèmes d'éducation à distance. Le langage peut en effet être utilisé pour décrire des activités pédagogiques dans un contexte plus traditionnel.

2.4.1.1 Méta-modèle de EML-OU

Le méta-modèle de EML-OU [Kop01] est constitué de quatre paquetages : *learning model*, *unit of study model*, *theories of learning and instruction* et *domain model*.

Learning model Ce modèle décrit comment l'apprenant apprend. Il se base notamment sur les axiomes suivants :

1. Une personne apprend en (inter-)agissant avec le monde extérieur.
2. Le monde réel peut être considéré comme composé de situations sociales et personnelles qui fournissent le contexte pour les actions.
3. Une situation est composée d'une collection de choses et de personnes dans le cadre d'une relation mutuelle.
4. Une partie des situations s'inscrit dans des communautés de pratique et plus particulièrement des communautés d'apprentissage.

Unit of study model Nous ne présenterons pas en détail le méta-modèle d'une unité d'étude car il est à la base de IMS-LD que nous verrons dans la section suivante. Nous pouvons dire qu'il se base sur le concept d'activités qui sont réalisées dans un environnement par des rôles.

Domain model Ce modèle décrit les caractéristiques spécifiques à un champ d'application (domaine bancaire, biologie, ...). Chaque modèle pédagogique doit prendre en compte les caractéristiques du domaine d'application. Les différents domaines ont leur propre structuration des connaissances et leur propre définition des compétences.

Theories of learning and instruction Ce modèle généralise les théories de l'apprentissage. Parmi elles on peut notamment citer l'approche empirique (connaissance=expérience), l'approche rationaliste (connaissance=pensée) et l'approche pragmatique (théorie de l'activité) (connaissance individuelle et de groupe). Le langage d'expression est indépendant d'une

théorie particulière. On retrouvera l'aspect théorique lors de la réalisation du modèle, i.e. son instantiation dans une plate-forme ou non. Il est en effet possible d'utiliser le modèle pour décrire un cours qui ne se déroule pas en e-learning.

Après avoir brièvement présenté EML-OU, nous allons maintenant nous intéresser à IMS Learning Design Specification qui s'appuie sur le modèle d'unité d'apprentissage de EML-OU.

2.4.2 IMS Learning Design Specification

Le travail réalisé par IMS dans le cadre du *Learning Design Specification* (IMS-LD) [IMS03c] s'appuie sur les travaux de l'Open University of the Netherlands portant sur EML (cf. section précédente) ainsi que sur les standards IMS existants et plus particulièrement ceux portant sur les méta-données (LOM) et sur l'empaquetage des contenus pédagogiques [IMS03a]. L'objectif est de fournir un cadre pour l'expression des activités pédagogiques et de promouvoir l'échange et l'interopérabilité. Nous n'allons pas revenir sur les motivations de EML-OU que nous avons présentées précédemment.

L'objectif de IMS-LD est de fournir une infrastructure d'éléments capables de décrire n'importe quelle conception de processus d'enseignement d'une manière formelle. Plus spécifiquement, IMS-LD remplit les caractéristiques suivantes :

Complétude. La spécification doit permettre de décrire complètement un processus d'enseignement dans une unité d'apprentissage en incluant des références à des contenus pédagogiques et à des services, tous deux pouvant être numériques ou non.

Flexibilité pédagogique. Il doit être possible d'exprimer le sens pédagogique des différents éléments contenus dans le contexte d'une unité d'apprentissage. La flexibilité se retrouve dans la capacité de réaliser les descriptions en utilisant différents types de pédagogies et en n'en prescrivant aucune en particulier.

Personnalisation. Cette spécification doit être capable de décrire des aspects personnalisés, afin que le contenu et les activités situés à l'intérieur d'une unité d'apprentissage puissent être adaptés en se basant sur les préférences, le dossier personnel, les connaissances antérieures, les besoins éducationnels et les usages contextuels des utilisateurs. De plus, le contrôle du processus d'adaptation doit être donné, selon le cas, à l'étudiant, un membre de l'équipe pédagogique, l'ordinateur et/ou un concepteur.

Formalisation. La spécification doit décrire un modèle d'apprentissage dans le contexte d'une unité d'apprentissage d'une manière formelle afin de permettre un traitement informatique du modèle.

Reproductibilité. La spécification doit décrire un modèle d'apprentissage abstrait de telle manière qu'il permette une répétition de l'exécution dans différentes configurations et avec différentes personnes.

Compatibilité. La spécification utilise les différents standards existants lorsque cela est possible (LOM, IMS-QTI).

Réutilisabilité. La spécification permet d'identifier, d'isoler, de dé-contextualiser et réutiliser les modèles dans d'autres contextes.

2.4.2.1 Méta-modèle

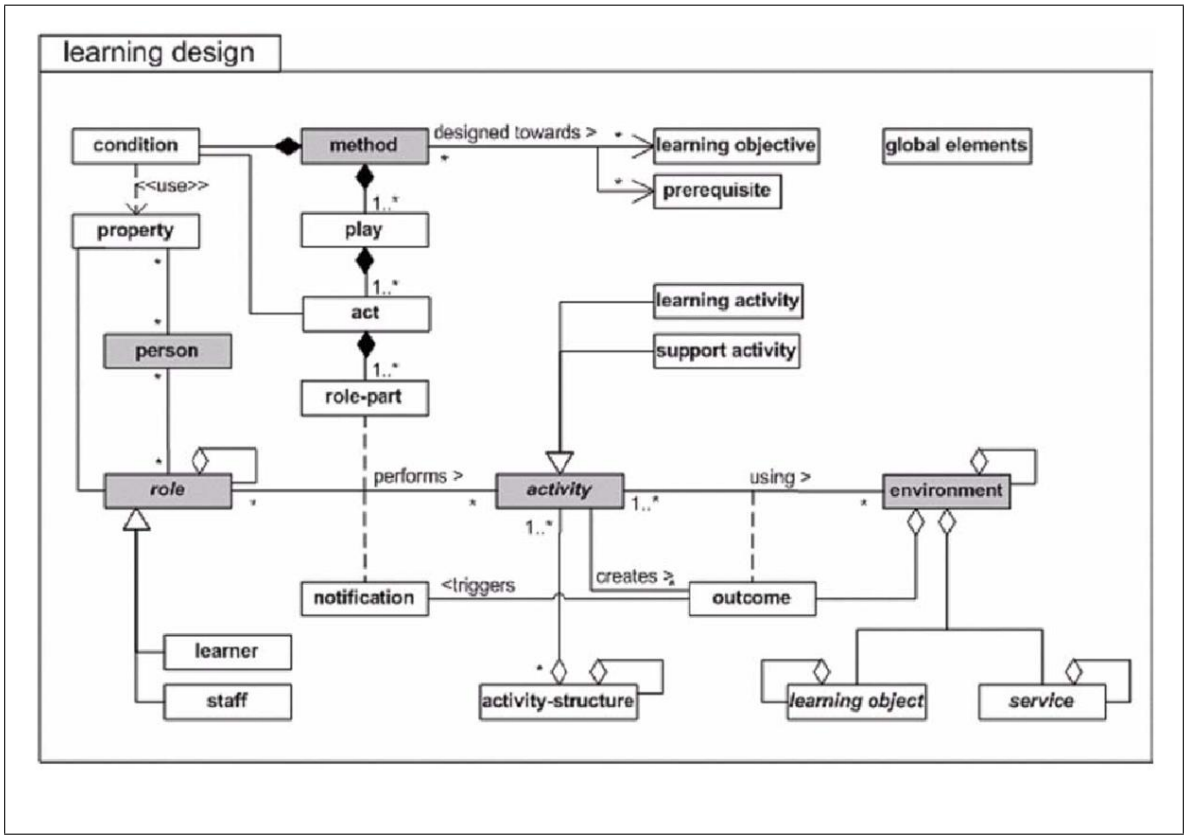


FIG. 2.7 – Méta-modèle de IMS-LD (tiré de [IMS03b])

Nous allons maintenant voir en détail le méta-modèle de IMS-LD, présenté à la figure 2.7. Les principaux éléments du méta-modèle sont *activity*, *role*, *environment* et *method*.

Les *activities* sont un des éléments centraux du « *learning workflow* ». Elles forment un lien entre les rôles et l'environnement d'apprentissage. Elles décrivent les activités qu'un rôle doit réaliser dans un environnement spécifique composé de *learning objects* et de *services*. Elles spécifient également leurs conditions d'arrêts et les actions à réaliser lorsqu'elles arrivent à leurs fins. Il existe deux types d'activités ; les activités d'apprentissage (*learning activities*) et les activités de support (*support activities*). Les activités d'apprentissage sont dirigées par des objectifs à atteindre. Elles sont constituées principalement d'une description du travail à réaliser et de liens avec un ou plusieurs environnements de travail. Les activités de support sont exclusivement réalisées par le rôle *staff*. Elles peuvent par exemple être des activités de correction de devoir ou d'aide pour des étudiants en difficultés. Une activité possède deux moyens principaux pour se terminer, soit l'utilisateur indique la fin de manière explicite soit la durée maximale pour l'exécution est atteinte.

Les *activities* peuvent être assemblées dans des *activity-structures*. Les structures d'activités agrègent un ensemble d'activités apparentées en une seule structure qui peut être associée

à un rôle dans un *role-part*. Une structure peut modéliser une séquence ou une sélection d'activités. Dans une *séquence*, un rôle doit réaliser complètement les différentes activités de la structure dans l'ordre indiqué. Dans une *selection*, un rôle peut sélectionner un certain nombre d'activités parmi l'ensemble fournit par la structure. Cela peut par exemple être utilisé pour la modélisation de situations où les étudiants doivent réaliser deux activités qui peuvent être choisies librement parmi un ensemble de cinq activités. Une structure d'activités peut également référencer d'autres structures d'activités ou des unités d'apprentissages externes, permettant ainsi l'élaboration de structures complexes multi-niveaux. Tout comme une activité, une structure d'activités peut référencer un ou plusieurs environnements. Cela permet de réaliser une série d'activités dans le même environnement.

Les *learning objectives* représentent les buts à atteindre par l'apprenant qui réalise les activités. Il est possible de spécifier les objectifs à deux niveaux, au niveau global de l'unité d'apprentissage ou au niveau de chaque activité. Les deux niveaux peuvent être utilisés en même temps. Le premier niveau définit les enjeux principaux de manière plus ou moins abstraite et le second niveau permet de situer précisément le travail à réaliser dans l'activité par rapport au but final.

Les *roles* explicites spécifiés dans ce langage sont ceux de *learner* et *staff*. Chacun d'eux peut être spécialisé en sous-rôles mais aucun vocabulaire n'est proposé. Cela est laissé ouvert aux concepteurs de nommer ces sous-rôles et de spécifier leurs activités. Par exemple dans le cas de jeux d'entreprise, différents étudiants peuvent jouer des rôles distincts, chacun réalisant ses propres activités dans des environnements différents. A l'exécution, un rôle peut être assigné à plusieurs *person*, il est cependant possible de placer des restrictions sur le nombre minimum et maximum de personnes associées à un rôle. Ce dernier permet alors de réaliser en quelque sorte des groupes.

L'*environment* contient une collection structurée de *learning objects* et de *services* appropriés qui vont être utilisés pour la réalisation des activités. Les *Learning objects* sont les différentes ressources numériques ou non, manipulées dans les activités. Elles sont généralement classées en utilisant les méta-données de LOM. On trouvera par exemple des pages web, des livres, des outils (traitement de texte, calculatrice, *etc*), des instruments (microscope, *etc*). Les *services* représentent des applications génériques communes à toutes les plates-formes comme les forums de discussions, les outils de communication synchrone (*chat*), les outils de recherche, *etc*. Ces services sont des ressources dont l'URL n'est pas donnée lors de la conception car cela obligerait l'utilisation des mêmes outils pour toutes les instances du modèle. Or certains outils demandent des configurations dynamiques comme par exemple un outil de discussion synchrone qui peut être réservé pour certain groupes d'utilisateurs qui ne sont connus que lors de l'instantiation. Pour régler ce problème la spécification propose une description précise de différents types de services indépendamment de leurs conditions d'utilisation. Ensuite, c'est la plate-forme d'exécution qui se chargera de mettre en œuvre les outils adéquats.

La *method* est conçue pour atteindre les *learning objectives* (spécification des résultats et/ou productions des étudiants), et présuppose certain *prerequisite* (spécification du niveau d'entrée des étudiants). La méthode consiste en une ou plusieurs occurrences de *play*. Elle se termine lorsque le *play* se termine ou lorsqu'une durée maximale est atteinte. Le *play* représente le processus d'apprentissage utilisant les éléments définis précédemment. Il est

modélisé selon une métaphore théatrale. Il consiste en une séquence d'un ou plusieurs *act* et un acte est relatif à un ou plusieurs *role-part* qui associent un seul rôle à une *activity* ou à une *activity-structure*. Les activités à l'intérieur d'un acte peuvent être réalisées en parallèle. Il est possible de spécifier des *conditions* sur l'exécution des actes. Un acte se termine lorsque tous les *role-part* sont terminés ou lorsque qu'une durée maximale est atteinte.

2.4.2.2 Conclusion

Cette spécification présente la forme la plus aboutie pour la description d'activités d'apprentissage de manière indépendante d'une pédagogie particulière. Elle se compose de quatres éléments principaux qui sont les activités (ce qui doit être fait), les rôles (représentation des personnes réalisant les activités), l'environnement (les ressources utilisées pour réaliser l'activité) et la méthode (ordonnancement entre les différentes activités).

Elle présente des points communs avec AICC dans le découpage des éléments en séparant le concept d'activité et l'ordonnancement de ces dernières. Mais contrairement à AICC il est possible de définir plus finement l'environnement dans lequel se situe l'activité.

D'un point de vue temporel, il est possible d'indiquer les durées maximales d'exécution des différentes activités ainsi que des différents éléments assurant leurs ordonnancement (*method*, *play* et *act*). Cette limitation évite aux étudiants de s'attarder dans des activités et les oblige à travailler de manière plus régulière. Mais elle n'est cependant pas très fine car il n'est pas possible de spécifier des temps minimaux évitant ainsi qu'un étudiant saute d'activité en activité sans réellement les étudier. Il n'est pas possible de spécifier des dates minimales et maximales d'exécution.

La notion de groupe est implicite. On n'exprime dans le modèle que des rôles exprimés sous forme d'identifiant pour permettre la réutilisabilité. Il est possible d'attribuer un même rôle à plusieurs personnes. C'est une limite pour nous qui avons une approche basée sur l'apprentissage collaboratif.

2.5 Conclusion du chapitre

Dans ce chapitre, nous nous sommes intéressés aux langages de modélisation pédagogique (EML) et nous avons dressé un panorama non exhaustif des langages existants en nous concentrant plus particulièrement sur leurs méta-modèles. Nous terminons ce chapitre en présentant le modèle de référence du CEN/ISSS WS/LT[3] et en comparant ce dernier aux différents EML que nous avons présentés. Nous conclurons enfin en déterminant, par rapport à nos critères, quel langage conviendra le mieux pour la suite de nos travaux.

2.5.1 Modèle de référence du CEN/ISSS WS/LT

Afin de comparer différents EMLs existants, le workshop sur les EMLs du CEN/ISSS a proposé un modèle de référence, présenté à la figure 2.8. Ce modèle présente l'intérêt de

[3]Comité Européen de Normalisation / Information Society Standardizaion System Workshop on Learning Technology

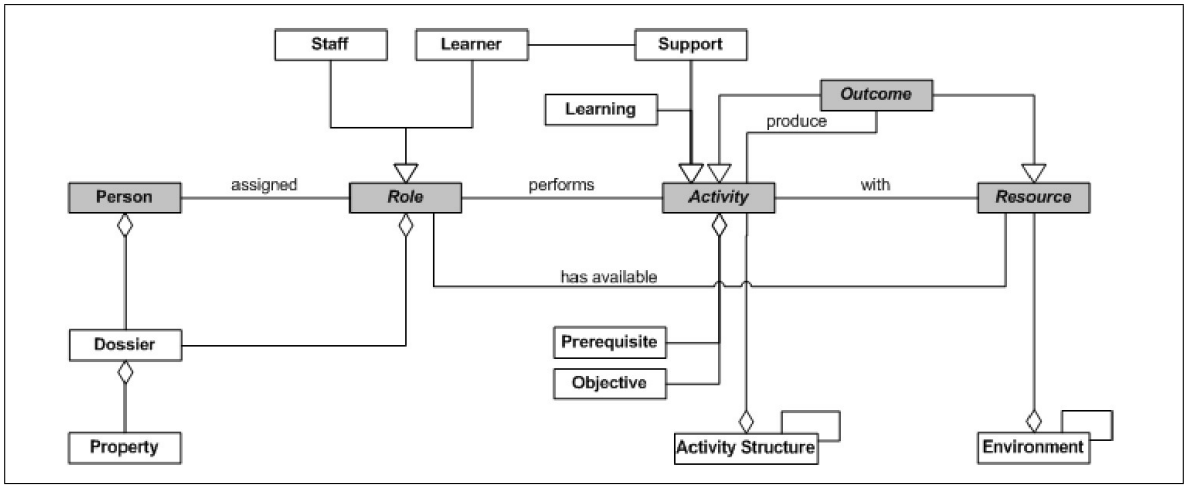


FIG. 2.8 – Méta-modèle basique d'un EML défini par le CEN/ISSS [RvRK+02]

s'intéresser à la modélisation pédagogique de manière générale. Il ne s'attarde pas sur un type d'activité particulier. L'activité est l'élément central. Elle représente le travail qui sera réalisé par des rôles. Les rôles sont assignés à des personnes. Une activité utilise un ensemble de ressources et en produit également.

Nous pouvons également remarquer des ressemblances avec la théorie de l'activité [BD00]. Nous avons le concept de rôle, d'activité, incluant la division du travail par l'intermédiaire de l'*Activity Structure*, la notion d'objet par l'intermédiaire des objectifs, de production et de ressources dans les deux cas.

Nous avons rapproché les entités définies dans les EML vus dans ce chapitre avec le modèle de référence du CEN/ISSS. Le tableau 2.1 présente une synthèse de ce travail qui permet de comparer les EML au modèle de référence et d'étudier la couverture des concepts nécessaires à la gestion des activités pédagogiques dans chaque EML. On peut ainsi constater que si tous les langages permettent de décrire une activité, seul IMS-LD permet de gérer les rôles et les acteurs. En étudiant ce comparatif, nous pouvons nous rendre compte que le modèle de référence s'apparente très fortement à celui de IMS-LD. La majorité des concepts sont disponibles dans les deux modèles et utilisent les mêmes termes.

2.5.2 Choix d'un langage

Dans le cadre de nos travaux, nous nous intéressons plus particulièrement à fournir un support d'exécution à des scénarios pédagogiques de manière la plus indépendante possible des langages existants car il n'existe pas de langage universel. Comme nous l'avons vu, chaque langage présente des intérêts et des particularités pour certains domaines (gestion des tests, structuration des ressources, ...). Le langage que nous allons choisir doit être le plus ouvert possible, i.e., s'intéresser à la description de tous types d'activités et ceci dans n'importe

CEN/ISSS	**TML**	**IMS-QTI**	**Targeteam**	**AICC**	**LMML** (Modèle didactique)	**IMS-LD**
Activity	Tutorial	Assessment / Section / Item	Issue	Assignable Unit	Learning Unit	Activity
Activity Structure	Tutorial / Question	Assessment / Section / Flow	Issue Group	Block	Learning Step	Activity Structure
Objective	-	-	Intro	Objective	Objective	Learning objective
Prerequisite	-	-	Intro	Prerequisite	Prerequisite	Prerequisite
Outcome	-	Outcomes	-	-	-	Outcome
Resource	Question	Material	Issue	Assignable Unit	Related	Learning Object / Services
Environment	-	-	-	-	-	Environment
Role	-	-	-	-	-	Role
Person	-	-	-	-	-	Person
Dossier	-	Examinee Record	-	-	-	-
Property	-	-	-	-	-	Property

TAB. 2.1 – Comparaison des différents EML

Langage	Temps	Structuration	Groupe
TML	-	Pas de structuration explicite	-
IMS QTI	Notion de durée	Structuration du flot de questions (séquentiel et parallèle)	-
Targeteam	-	Structuration des ressources (séquentiel)	-
AICC	-	Structuration des leçons (séquentiel et parallèle)	-
LMML	-	Structuration des leçons (séquentiel)	-
IMS LD	Durée d'une activité	Structuration des activités et de leurs exécution (séquentiel et parallèle)	Par l'intermédiaire des rôles

TAB. 2.2 – Comparaison des EMLs par rapport aux notions de temps, de structuration et de groupe

quel contexte. Il devra également s'approcher le plus possible des standards existants et de nos préoccupations. Afin de vérifier l'adéquation des langages avec nos préoccupations, nous avons dressé un tableau comparatif (tableau 2.2). Il nous permet de voir les différentes notions temporelles supportées par le langage, les types de structuration et d'ordonnancement des activités et les différents aspects relatifs à la gestion de groupe. Il en ressort qu'aucun langage ne remplit l'intégralité de nos critères. Celui qui s'approche le plus est IMS-LD car il permet de gérer les durées d'exécution et les groupes (toutefois sans offrir facilement la possibilité de définir des parcours individuels à l'intérieur d'un groupe). Si nous reprenons le tableau comparatif 2.1, nous voyons qu'il est le langage s'approchant le plus du modèle de référence. Son niveau de description est assez abstrait, généraliste et indépendant du modèle pédagogique. De plus, il pourrait très vite devenir un standard de facto pour la description des modules pédagogiques. Il s'intéresse à la modélisation de l'activité, ce qui est un des fondements des recherches récentes de notre laboratoire [Bou00]. En conséquence, nous avons choisi d'utiliser IMS-LD comme langage de description des scénarios pédagogiques afin de valider notre démarche qui consiste à montrer la possibilité d'utiliser un EML sur un moteur de workflows, et le travail que cela nécessite. Nous allons nous baser sur un seul langage en montrant clairement les différentes phases de transformation vers un langage généraliste de description de processus afin de pouvoir réitérer l'opération pour d'autres EMLs.

Chapitre 3

Les systèmes de workflows

> There are two major products that come out of Berkeley : LSD and UNIX. We don't believe this to be a coincidence.
>
> – **Jeremy S. Anderson**

Dans le chapitre précédent, nous avons présenté quelques uns des EML existants. Nous avons notamment pu remarquer que l'aspect coordination entre activités était un élément important (par exemple dans IMS-LD, AICC). Dans le domaine administratif, la coordination des tâches à réaliser est souvent assurée par un système de workflows. Ce chapitre a pour objectif de présenter de manière globale les systèmes de workflows et leurs différentes caractérisques. Nous commencerons par présenter l'origine et la définition des systèmes de workflows dans la section 3.1. Nous verrons ensuite les principaux standards utilisables pour la conception et l'intégration d'un système de workflows (section 3.2). Cela nous servira de point de départ pour notre implémentation. Nous présenterons dans la section 3.3 des caractéristiques particulières des systèmes de workflows qui nous semblent intéressantes dans le cadre de nos travaux telle que la gestion de la flexibilité et nous verrons quelques exemples de mise en œuvre dans la section 3.4. Enfin la section 3.5 concluera ce chapitre.

3.1 Présentation des systèmes de workflows

3.1.1 Origines et évolution

La gestion du flux de travail se retrouve dans de nombreux domaines différents comme par exemple la construction mécanique, l'ingénierie logicielle [JBR00], le commerce électronique [LSAS01], la gestion électronique de documents et le travail coopératif assisté par ordinateur. L'origine des systèmes de workflows remonte à la fin des années 1970 et aux différents travaux relatifs à l'automatisation des tâches administratives afin d'en diminuer le temps de traitement

[Zis77]. Les procédures étaient formalisées et un système informatique assurait la répartition des tâches et la circulation des documents. Ces premiers systèmes furent mal accueillis par les utilisateurs pour deux raisons principales : l'une technique liée notamment au manque d'infrastructure réseau qui limitait ainsi la circulation des documents, et l'autre humaine car les procédures étaient jusque là considérées comme laissant une grande place à des activités non complètement structurées. Bien que similaire, deux instances de la même procédure ne sont pas rigoureusement identiques (variation des documents traités par exemple). Les systèmes de cette époque étaient très rigides et n'autorisaient pas de modification du modèle en cours d'exécution. Des recherches ont été ensuite menées sur la modélisation des procédures comme par exemple les travaux de Winograd et Flores [WF86] qui débouchèrent sur la conception du système *The Coordinator* utilisant le modèle de conversation basé sur les actes de langages [Sea69]. Depuis le début des années 1990, de nombreux travaux se sont focalisés sur la capacité des systèmes à pouvoir adapter et modifier le modèle de processus dynamiquement en cours d'exécution [GAC+97, WID03, LASS00, EGL98, Man00] ou sur la capacité à réagir face à des exceptions [HA98].

3.1.2 Définitions

Les définitions que nous utilisons proviennent du glossaire [Wor99b] réalisé par le Workflow Management Coalition (WfMC) [WfM03], organisme dont nous parlerons plus en détail dans la section 3.2.1. Le **workflow** est un système qui s'occupe de l'automatisation de procédures durant lesquelles des documents, des informations ou des tâches sont traités par des participants selon un ensemble de règles afin d'accomplir ou de contribuer à la réalisation d'un objectif global. Les participants peuvent être à la fois humain et non-humain (par exemple, traitement automatique d'un document réalisé par un ordinateur). Une **procédure** est un ensemble coordonné d'activités qui sont reliées afin d'atteindre un objectif commun. Une **activité** est la description d'un travail qui forme une étape logique à l'intérieur d'une procédure. Quelques autres définitions sont présentées dans le tableau 3.1.

3.1.3 Caractéristiques

Le cadre des systèmes de workflows est composé de trois principaux éléments. La figure 3.1 les représente ainsi que leurs relations :

Les fonctions de conception. Elles correspondent à l'analyse des différentes procédures et à leurs formalisations dans une forme manipulable par un système de gestion de workflows. L'objectif est de déterminer, avec plus ou moins de précision selon le système visé, les différentes activités, leurs relations (le flux de contrôle) ainsi que leurs paramètres (conditions de départ et d'arrêt, liste des participants, documents en entrée et en sortie).

Les fonctions de contrôle de l'exécution. Elles correspondent à l'interprétation des différents modèles de procédures et à la création de nouvelles instances. Elles gérent l'ordonnancement des activités et les ressources. Le composant de base est le moteur de workflows.

Vocabulaire	Définition
Procédure (Process)	Ensemble coordonné d'actions qui sont reliées dans le but d'atteindre un objectif commun.
Workflow	Automatisation des procédures durant lesquelles des documents, des informations ou des tâches sont échangés entre les participants.
Système de gestion de workflow (Workflow Management System)	Système qui définit, crée, gère et exécute les procédures de workflow.
Moteur de workflow (Workflow Engine)	Un service logiciel qui fournit l'environnement d'exécution pour une instance de procédure.
Interopérabilité des workflows (Workflow Interoperability)	La capacité pour des moteurs de workflows de communiquer et de travailler ensemble pour coordonner leur travail.
Service de promulgation du Workflow (Workflow Enactement Service)	Un service logiciel qui consiste en un ou plusieurs moteurs de workflow afin de créer, gérer et exécuter différentes instances de workflow.
Activité (Activity)	Description d'une action qui forme une étape logique à l'intérieur d'une procédure
WAPI (**W**orkflow **API**s and **I**nterchange Formats)	WAPI est utilisé pour permettre l'interopérabilité entre les composants d'un système de gestion de workflow et des applications externes.

TAB. 3.1 – Quelques définitions du glossaire du WfMC ([Wor99b]).

Les interactions lors de l'exécution. Les différentes activités d'un processus requièrent généralement des opérations humaines, souvent en coopération avec des outils informatiques pour traiter des données. Les résultats de ces traitements manuels ou automatiques nécessitent une communication avec les fonctions de contrôle de l'exécution, notamment pour préciser au moteur qu'une activité est terminée et fournir les documents résultants.

Après avoir montré les trois principales fonctions d'un système de workflows, nous allons maintenant plonger plus en détail dans l'implémentation de ces systèmes.

3.1.4 Modèle d'implémentation

Malgré la variété des produits existants, il est possible de dresser un modèle générique de l'implémentation des workflows. Ces principales fonctions sont reprises dans la figure 3.2 tirée de [Wor95]. Le double intérêt de ce modèle réside dans la présentation claire des interfaces

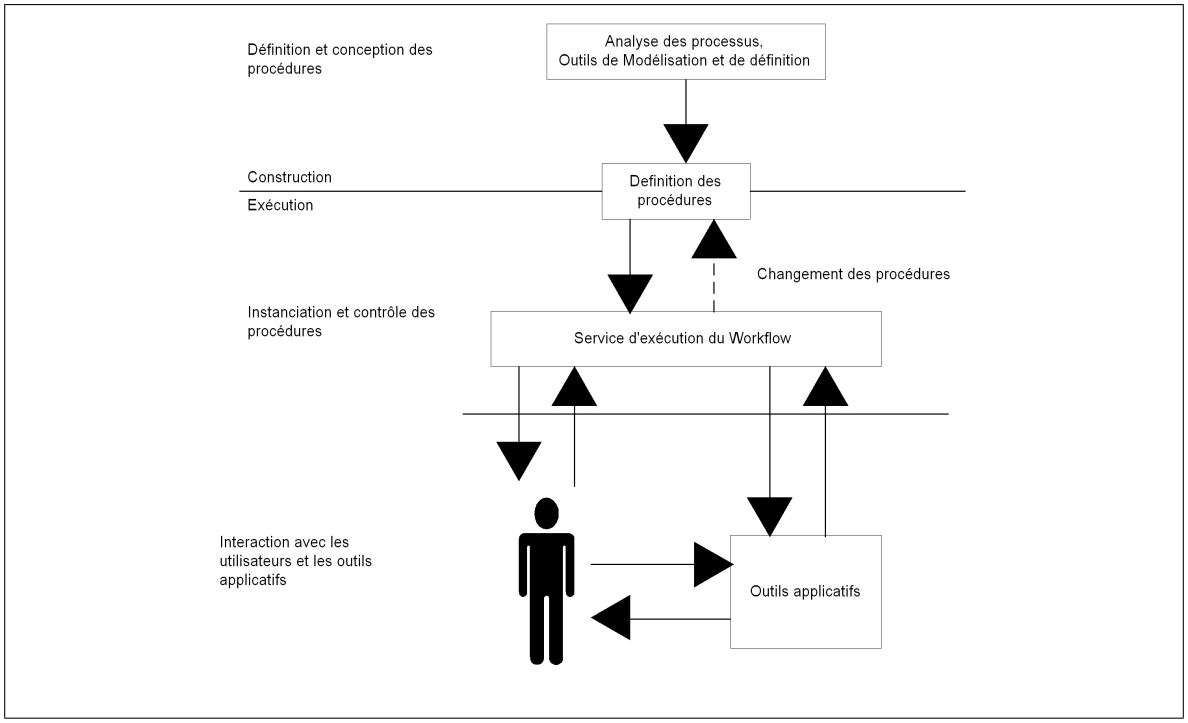


FIG. 3.1 – Caractéristiques d'un système de workflow, adapté de [Wor95]

communes à chaque système qui pourront faire l'objet de standardisation mais il nous offre également une vision globale pour la conception et le développement de notre propre système dédié à l'éducation à distance. On distingue cinq éléments principaux qui sont :

La définition du modèle. Comme nous l'avons dit précédemment, les workflows sont utilisés dans de très nombreux domaines mais généralement les moteurs d'exécution sont les mêmes. Ce qui différencie le plus les domaines ce sont les méthodes d'analyse et de conception des modèles qui manipulent des concepts très divers. Par exemple, un ingénieur pédagogique manipulera des modules de formations constitués d'activités répondants à un objectif pédagogique alors qu'un architecte logiciel manipulera des phases de développement constituées d'itérations. Les outils de manipulation et de création des modèles seront donc très différents car destinés à des publics aux compétences diverses et manipulant des concepts variés. Cependant, la majorité de ces modélisations repose sur des concepts communs de structures ordonnées d'activités. Il apparaît alors le besoin d'un langage permettant l'expression des modèles et leurs échanges entre différentes plates-formes. Ces modèles peuvent faire des références à des applications externes et à certains rôles du modèle organisationnel. Le lien lors de l'exécution entre rôle et personne physique est laissé à la charge du service d'exécution.

Le service d'exécution du workflow. Ce service est le cœur de la plate-forme de workflow. Il va interpréter les modèles et gérer les différentes instances. Il est principalement constitué de deux composants : le moteur de workflows et un gestionnaire des tâches. Le

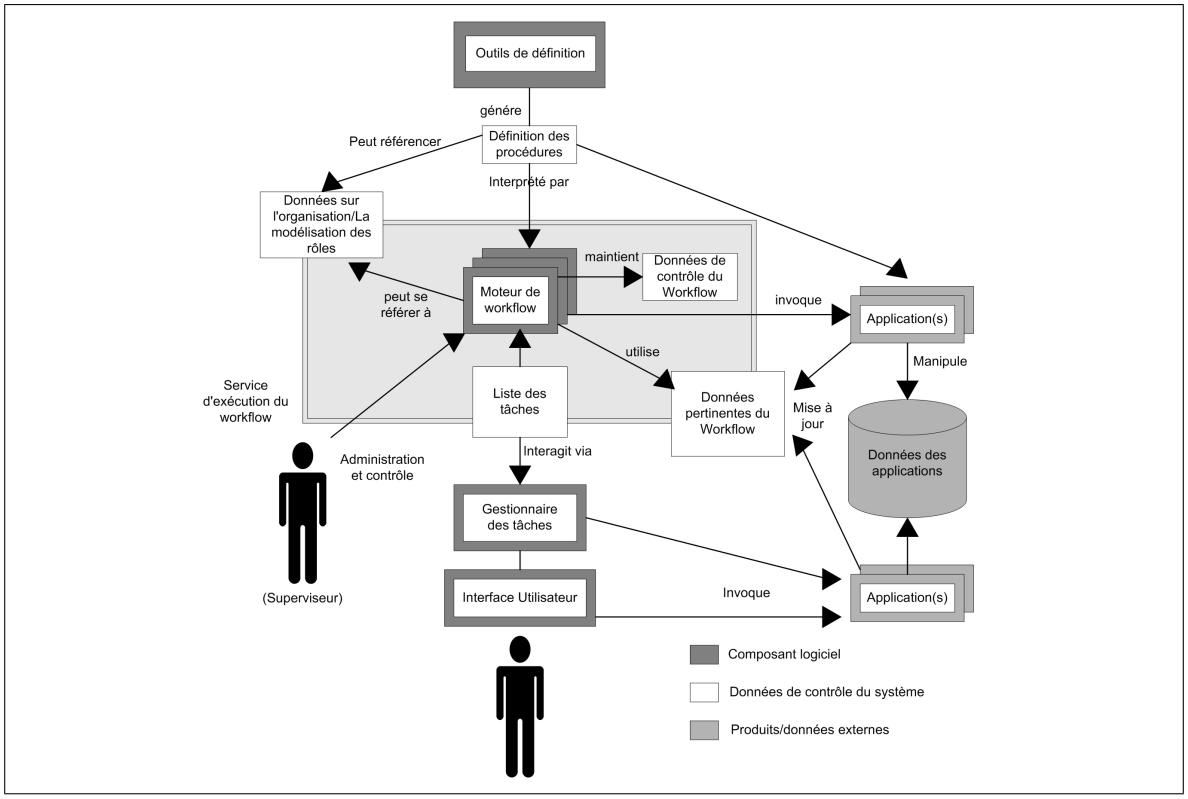


FIG. 3.2 – Structure générale d'un système de workflow [Wor95]

moteur de workflows va assurer le bon déroulement des processus, la coordination des activités en fonction des différentes règles de gestion du modèle et des différentes données manipulées. Il assignera via le gestionnaire de tâches les activités aux participants humains ou aux applications externes.

Les applications externes. Ce sont des outils utilisables par les participants humains pour la réalisation de leurs activités ou des applications directement contrôlées par le moteur. Il apparaît le besoin d'avoir une interface standardisée du moteur de workflows afin de permettre aux différentes applications externes de dialoguer avec lui.

La gestion des utilisateurs. Ces services permettent d'avertir les utilisateurs humains des tâches qu'ils ont à réaliser et de donner accès aux outils qui y sont associés. Ils permettent également aux participants de prévenir le moteur de la réalisation de leur travail.

L'administration du système. La gestion des différents paramètres d'un workflow comme le suivi des activités demande des outils de surveillance et d'administration adaptés au contexte de processus qui peuvent avoir une très longue durée. Une des missions de l'administrateur sera notamment de résoudre les problèmes. Cette mission d'administration peut comporter la surveillance des différents systèmes de workflows d'une entreprise. Il apparaît le besoin d'une interface standardisée permettant à un outil d'administration de surveiller plusieurs moteurs de workflows.

Tous ces éléments vont être utilisés pour définir le modèle de référence des workflows présenté dans la section 3.2.1.

3.2 Les principaux standards

Dans cette section, nous allons voir les principaux efforts réalisés dans le cadre de la standardisation et de l'intéropérabilité pour les systèmes de workflows. Nous commencerons par nous intéresser au *Workflow Management Coalition* (WfMC) et à ses différents travaux de standardisation puis nous verrons la proposition de l'OMG pour la standardisation des interfaces d'un moteur de workflows.

3.2.1 La proposition du Workflow Management Coalition

Le Workflow Management Coalition, fondé en août 1993, est le principal promoteur de la standardisation des systèmes de workflows. Cette coalition, composée à la fois de vendeurs, d'utilisateurs et de laboratoires de recherche, vise la promotion et le développement des workflows par la mise en place de standards d'intéropérabilité et de connexion.

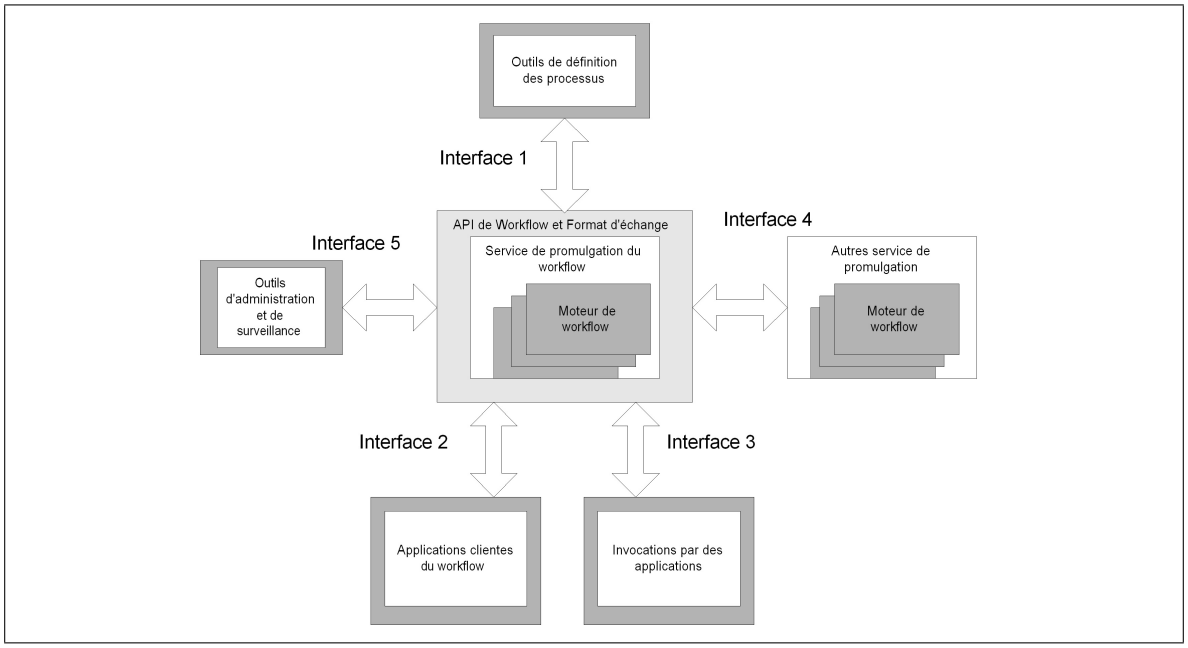


FIG. 3.3 – Modèle de Référence du Workflow

Pour la réalisation des différentes spécifications, les membres du WfMC ont tout d'abord réalisé un modèle de référence (figure 3.3) reprenant l'ensemble des éléments communs à tous les systèmes de workflow. Ce modèle, décrit dans le document [Wor95], sert de cadre pour la réalisation de toutes les spécifications. Nous allons dans les paragraphes suivants détailler un peu plus les différentes interfaces de standardisation.

3.2.1.1 Définition des processus (interface 1)

Comme nous l'avons déjà indiqué, les systèmes de workflows sont utilisés dans des domaines différents. Cela signifie en particulier que la modélisation des processus fait appel à des experts du domaine. Un ensemble d'outils différents peut être utilisé pour analyser et définir les procédures d'un workflow. Cependant, le modèle du workflow est indépendant de la nature des divers outils. Le résultat doit être standardisé afin de pouvoir interconnecter n'importe quel outil de modélisation avec n'importe quel moteur.

Le WfMC a donc établi un langage appelé WPDL (Workflow Process Definition Language) [Wor99a] permettant l'échange de modèles entre les outils de modélisation et les services de promulgation des workflows.

La figure 3.4 représente l'échange du modèle entre les outils de modélisation et le service de workflow.

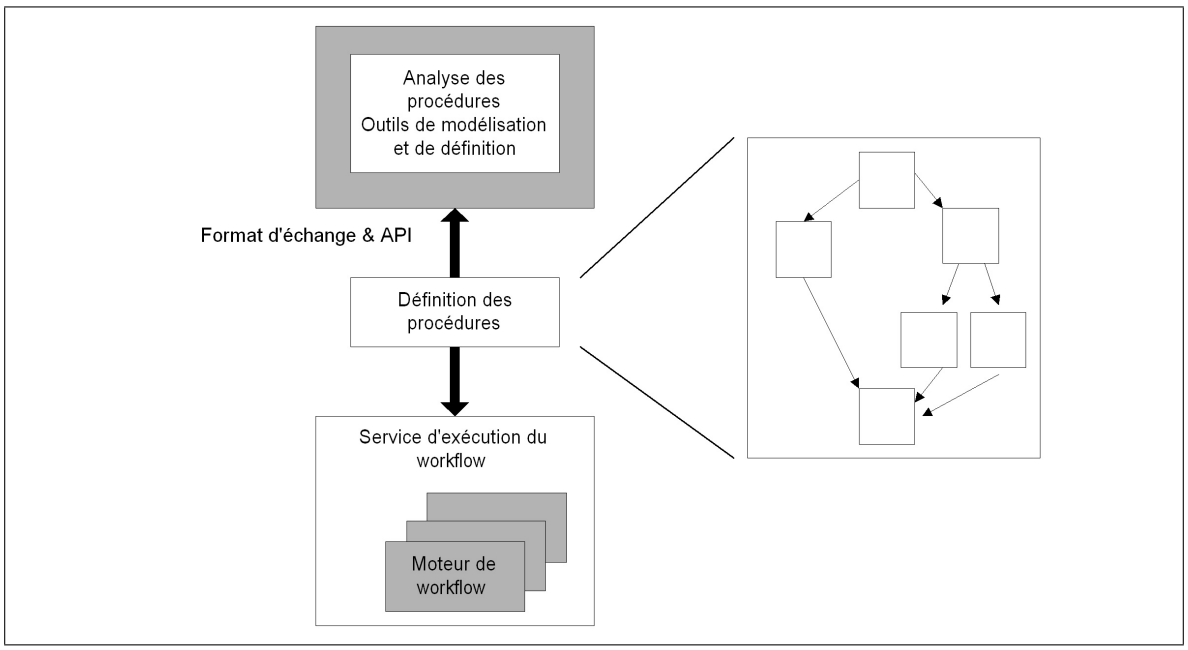


FIG. 3.4 – Échange de définition de processus

Afin de tenir compte de la part de plus en plus importante faite à XML, WPDL a évolué vers *XML Process Definition Language* (XPDL) [Wor02], dont nous allons voir plus en détail le métamodèle (figure 3.5). Il est composé de 5 entités principales qui sont Workflow Process Definition, Workflow Process Activity, Transition, Participant et Application.

Workflow Process Definition. L'entité de définition de processus contient des informations contextuelles qui s'appliquent aux autres entités du processus. Il définit l'objectif global du travail à réaliser. Il fournit des informations d'ordre administratif (date de création, auteur, *etc*) ou utilisées au cours de l'exécution (paramètres d'initialisation, limites temporelles, *etc*). Il se compose d'une ou plusieurs activités.

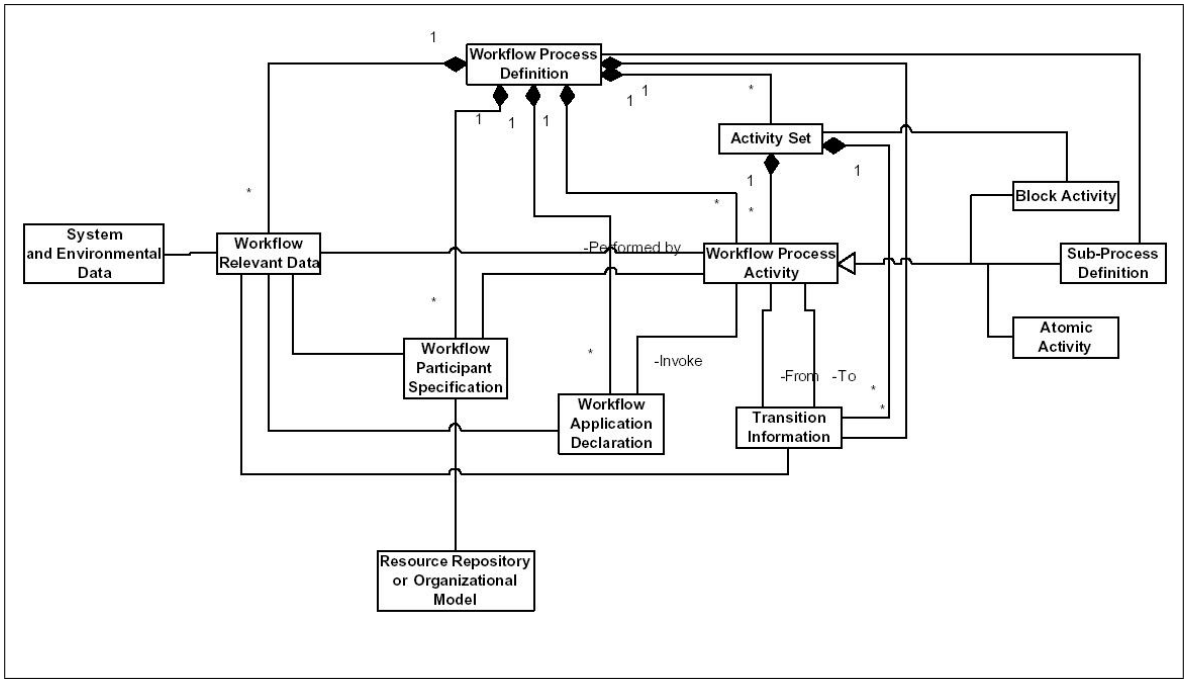


FIG. 3.5 – Méta-modèle de XPDL (tiré de [Wor02])

Workflow Process Activity. L'entité activité définit le travail qui sera réalisé par un ensemble de ressources (participants, applications informatiques, *etc*). Certaines informations peuvent être associées comme par exemple la priorité, le type de démarrage et d'arrêt (automatique ou manuel), *etc.* Il existe trois catégories d'activités. Une activité peut être un sous-processus (*Sub-process definition*), i.e., elle est un conteneur pour l'exécution d'un processus spécifié séparément. Ce processus peut s'exécuter localement ou à distance. Une activité peut également être une activité de bloc (*bloc activity*), i.e., l'exécution d'un regroupement d'activités et de transitions (*activity set*). Enfin, une activité peut être atomique (*atomic activity*). Il peut s'agir soit d'une activité de routage qui ne réalise aucun travail mais qui est utilisée pour synchroniser un ensemble de transitions, soit d'une activité « normale ». Dans ce cas lors de l'exécution, cette activité est transformée en bons de travail (*workitem*). Les activités sont reliées entre elles via des transitions.

Transition Information. Ces entités représentent le flot de contrôle, i.e., l'enchaînement des différentes activités. Chaque transition possède trois caractéristiques principales, le *from-activity* (l'activité initiale), le *to-activity* (l'activité d'arrivée) et les conditions de passage d'une activité à l'autre.

Workflow Participant Specification. Cette entité fournit la description des ressources qui vont réaliser les activités. Il existe plusieurs types de participants (*human*, *role*, *system*, *organizational unit*).

Workflow Application Declaration. Cette entité fournit des informations sur les applications informatiques qui peuvent être invoquées par le moteur de workflows afin de réaliser tout ou partie du traitement associé à une activité.

Workflow Relevant Data. Cette entité représente les données créées et utilisées dans chaque instance de processus. Ces données sont disponibles pour les différentes activités, transitions (pour l'évaluation de conditions par exemple) ou applications (paramètres d'entrée et de sortie).

Des outils open-source commencent à apparaître pour la réalisation des modèles en XPDL comme par exemple JaWE (Java Workflow Editor) [Enh03], présenté à la figure 3.6.

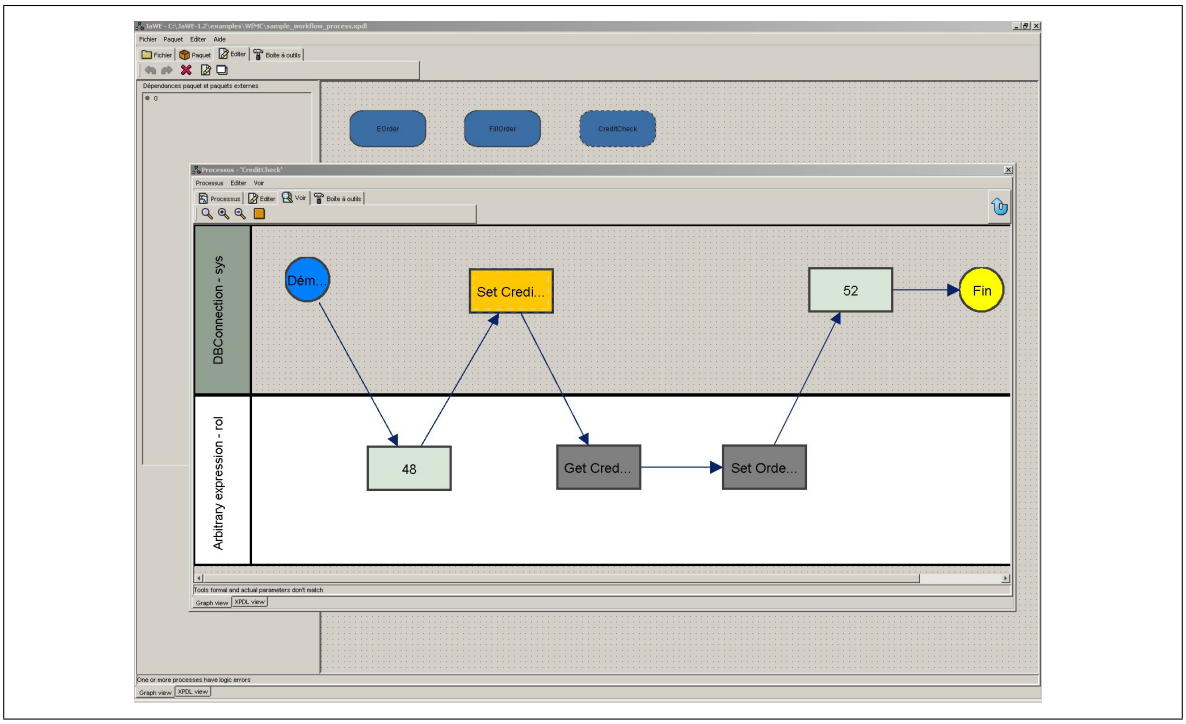


FIG. 3.6 – JaWE

Ce langage est relativement ouvert car il permet la définition d'éléments spécifiques par l'intermédiaire d'une entité *extended attributes*. Cependant, l'utilisation abondante de tels éléments peut causer des problèmes lors de l'échange de modèles, notamment si la majorité de la sémantique est contenue dans de telles balises.

3.2.1.2 Interface des applications clientes (interface 2)

L'objectif de cette interface est de fournir un ensemble standardisé d'API[4] permettant à des applications clientes d'accéder au moteur de workflows [Wor98b]. L'API proposée couvre

[4]API : Application Programming Interface

des domaines comme l'établissement de session, des fonctions de contrôle de procédures, des fonctions de supervision (figure 3.7).

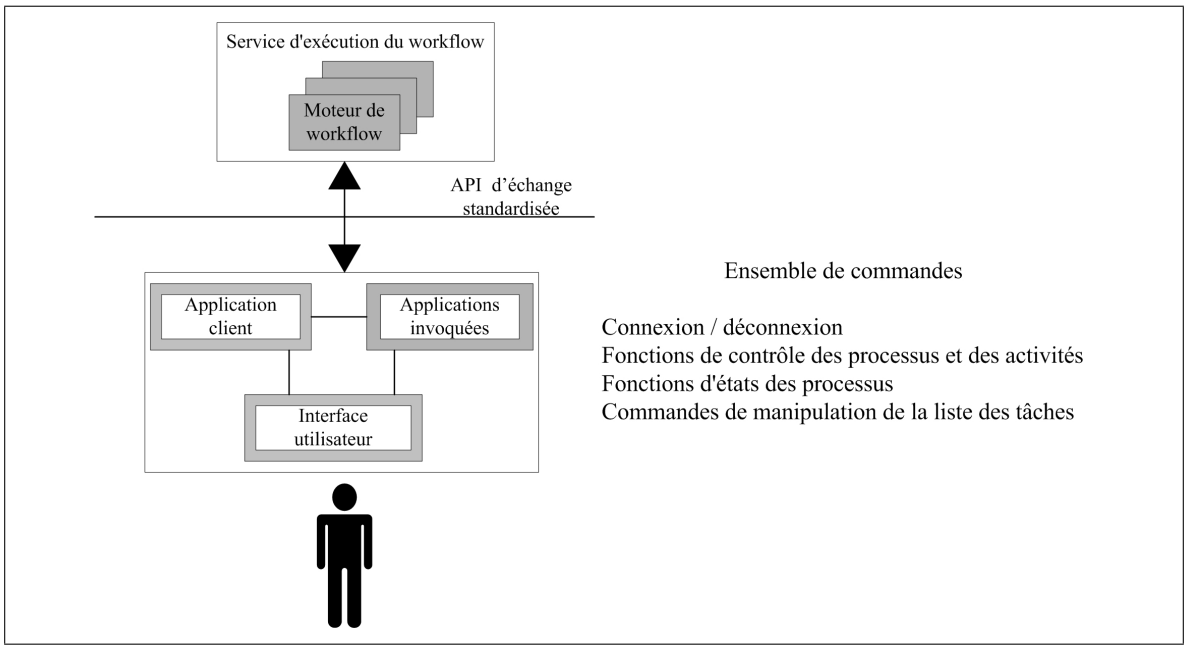


FIG. 3.7 – Interface pour les applications clientes

L'interface entre clients et service d'exécution doit comporter un identifiant de l'instance de procédure et de l'activité, les noms et adresses des ressources, des structures de données et des mécanismes de communication. Le WfMC propose notamment une version de son API en utilisant des interfaces IDL d'objets CORBA.

3.2.1.3 Interface d'invocation d'applications (interface 3)

L'interface d'invocation d'applications reprend certains éléments de l'interface 2 (méthodes communes, conventions de nommage, type de données, ...). Ces deux interfaces sont d'ailleurs réunies dans le même document de spécification [Wor98b]. L'objectif est de fournir une API permettant au moteur de workflow de faire appel à des applications externes. Il peut ainsi démarrer des programmes ou les interroger pour connaître leurs états. Cela permet notamment de suivre l'état d'avancement d'une activité en connaissant les résultats intermédiaires de l'application. Cette interface définit le concept de *Tool agent* dont l'objectif est de fournir un système, avec une interface spécifiée, encapsulant et gérant un ou plusieurs outils ou catégorie d'outils (figure 3.8).

3.2.1.4 Interopérabilité entre workflows (interface 4)

Un des objectifs clé de la WfMC est de définir des standards permettant à différents systèmes de workflows produits par différents vendeurs de travailler ensemble et de s'échanger

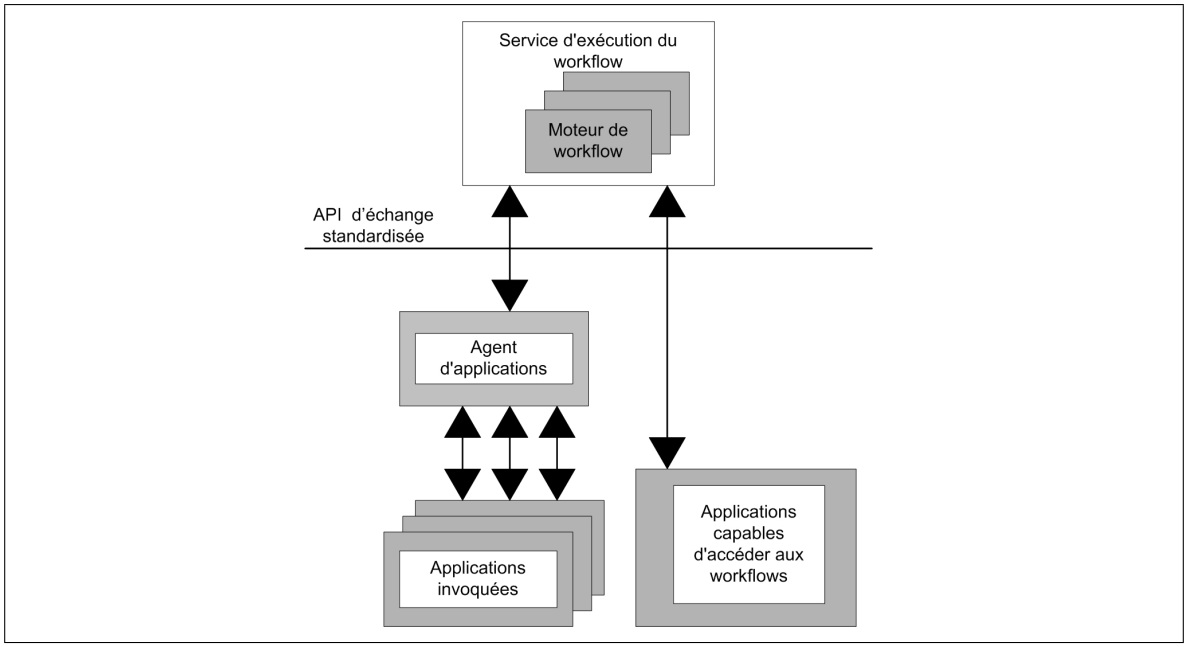


FIG. 3.8 – Interface pour l'invocation d'applications

des données de manière transparente. Les principaux échanges de données sont représentés dans la figure 3.9.

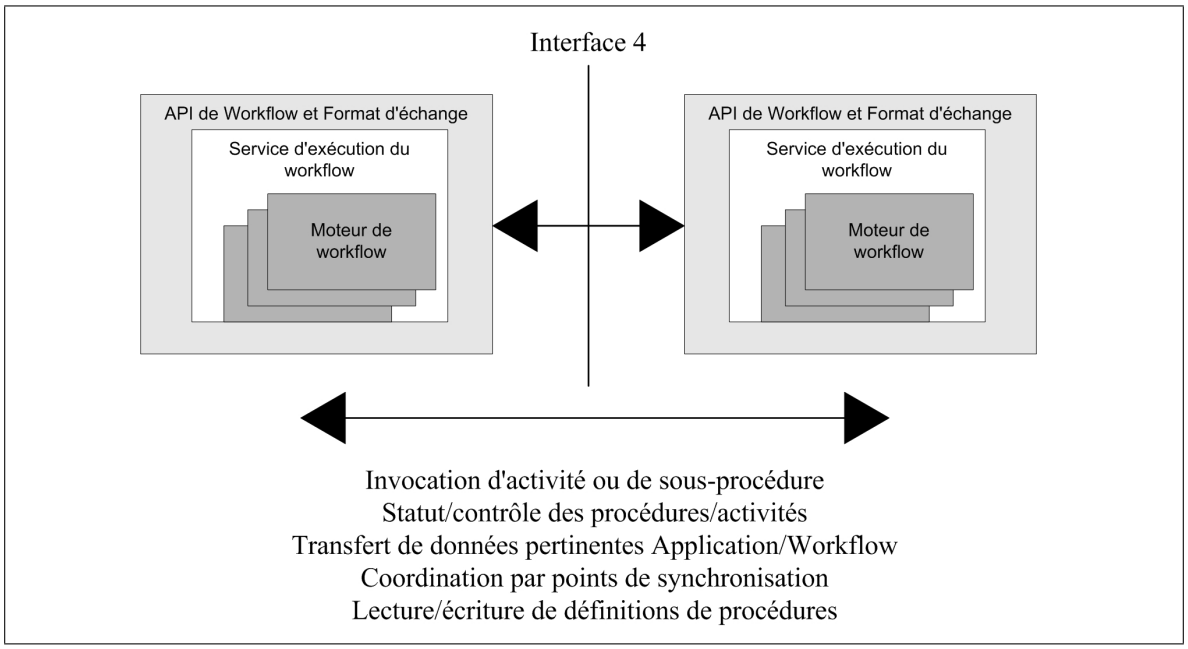


FIG. 3.9 – Interface d'interopérabilité entre workflows

Pour permettre les échanges de données entre moteurs, le WfMC a conçu une représentation XML de ces données. Ce langage, appelé Wf-XML [Wor01], facilite la communication entre systèmes et plates-formes hétérogènes. L'envoi et la réception des messages Wf-XML reposent sur des protocoles standards de l'Internet (HTTP et SMTP). Cette interface est de plus en plus implémentée dans les produits commerciaux.

Elle présente selon nous au moins un inconvénient majeur, elle ne permet pas obligatoirement d'avoir une vision complète du processus global. En effet, il est possible de lancer à distance un processus sans connaître son fonctionnement interne (principe de la boîte noire comme en programmation). Cette représentation incomplète est gênante dans le cas de l'éducation à distance où l'on souhaite fournir une vue globale du chemin d'apprentissage.

3.2.1.5 Administration du système (interface 5)

Cette spécification [Wor98a] propose une interface standard pour un ensemble de fonctions de gestion et d'administration afin de permettre à un vendeur d'application de gestion de travailler avec les moteurs de workflow d'autres vendeurs. Cette interface permet à différents services du workflow de partager un ensemble de fonctions communes d'administration.

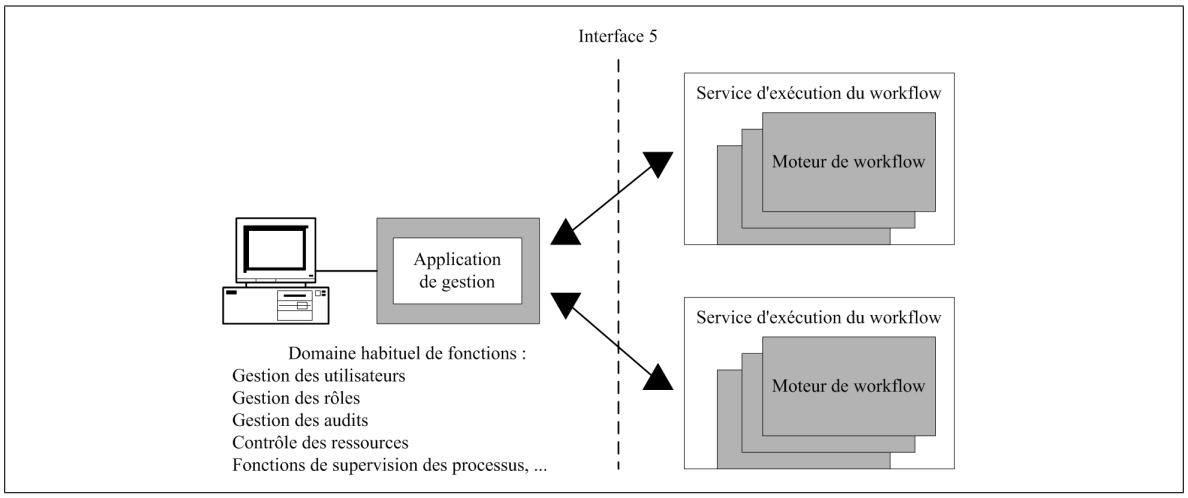


FIG. 3.10 – Interface de gestion

La figure 3.10 permet d'illustrer ces principes. Une application de gestion indépendante interagit avec deux moteurs de workflow différents. Chaque métier peut disposer d'un outil de visualisation adapté à ses besoins. Un analyste commercial aura une vue différente d'un administrateur de workflows mais ils manipulent les mêmes données. Cette interface est assez peu implémentée dans les produits actuels.

3.2.2 La proposition de l'OMG

L'Object Management Group (OMG) s'est basé sur le travail du WfMC et son modèle de référence pour proposer le Workflow Management Facility (WMF), un standard pour repré-

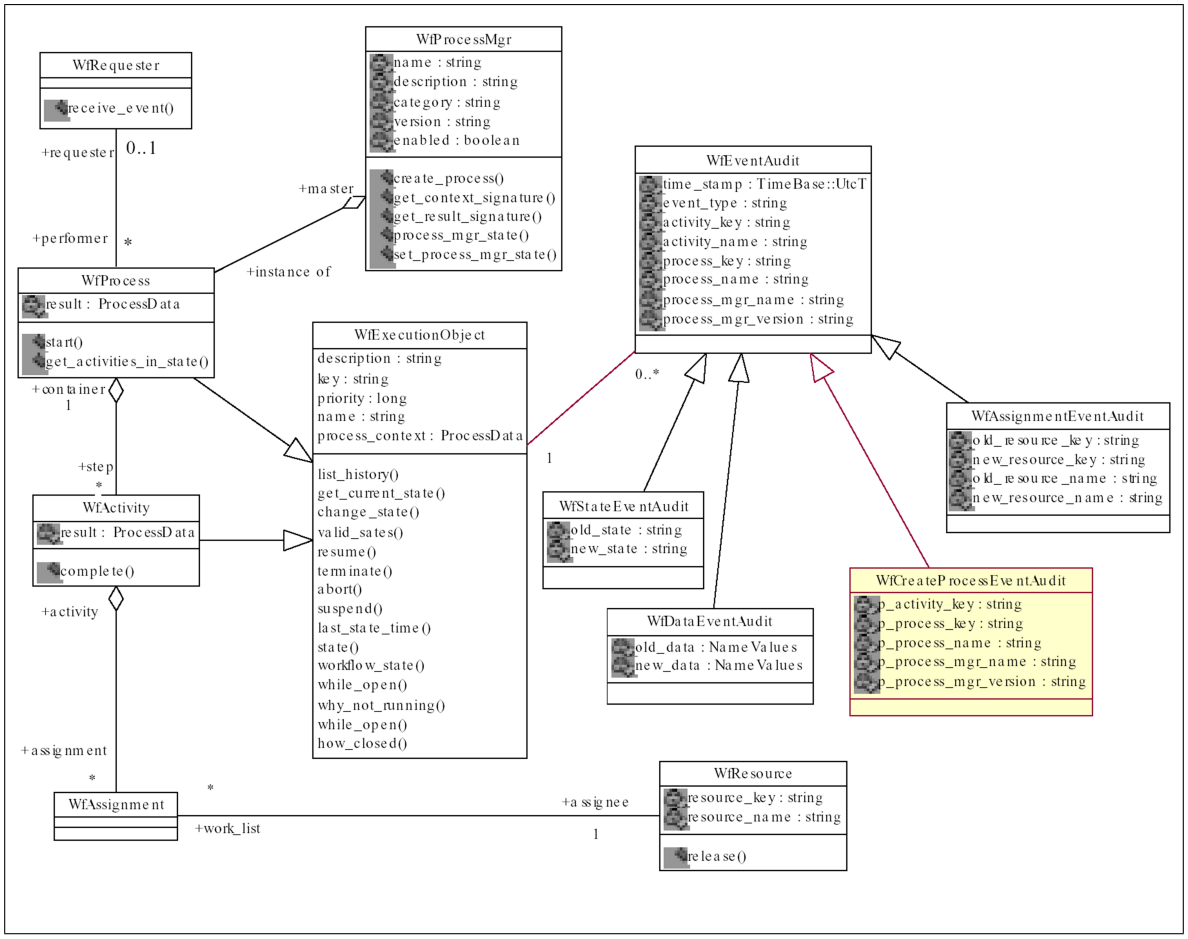


FIG. 3.11 – La spécification WMF de l'OMG (tiré de [OMG00b])

senter de manière objet les différents constituants d'un moteur de workflows. Les principaux objets de cette spécification sont représentés par le diagramme UML de la figure 3.11.

Un objet `WfProcess` représente une instance de processus. Il est rattaché à un objet `WfProcessMgr` qui représente un modèle de processus (i.e. il y a autant de `WfProcessMgr` que de modèles de workflows utilisables). C'est lui qui va permettre la création d'une nouvelle instance de `WfProcess`. Il fournit les méta-informations associées au contexte d'un processus, i.e., les paramètres d'entrée et de sortie. Un gestionnaire de processus est identifié par son nom qui doit être unique à l'intérieur d'un domaine. Il peut être retrouvé via le service de nommage, le service de courtage ou d'autres infrastructures. Un `WfProcess` gère les instances de `WfActivity`. `WfProcess` et `WfActivity` héritent de `WfExecutionObject`. Une activité est réalisée par une `WfResource` qui représente tout élément pouvant prendre part à la résolution d'une activité, i.e. un être humain, un logiciel, un robot, *etc.* Une activité est liée à une ressource par un `WfAssignment`. Cet objet vérifie que la ressource est disponible avant de l'allouer à une activité.

Il existe également une aborescence d'objets pour enregistrer les différents événements qui se produisent au cœur du moteur. L'interface *WfEventAudit* est utilisée pour définir l'interface de base. Elle fournit les informations sur la source et les données spécifiques à l'événement. Elle peut être étendue à loisir par les développeurs de la spécification par spécialisation de `WfEventAudit`, par exemple pour surveiller la création d'un processus (*WfCreateProcessEventAudit*), le changement d'état (*WfStateEventAudit*), le changement de valeur d'une donnée (*WfDataEventAudit*) et l'assignation des ressources (*WfAssignmentEventAudit*). Les événements sont persistants, et leurs durées de vie ne sont pas limitées par la durée de vie de l'objet qui les a produits. Ils peuvent être utilisés pour réaliser une évaluation a posteriori du comportement du moteur afin d'apporter par exemple une amélioration du modèle de processus.

3.2.2.1 Diagramme d'états des processus et activités

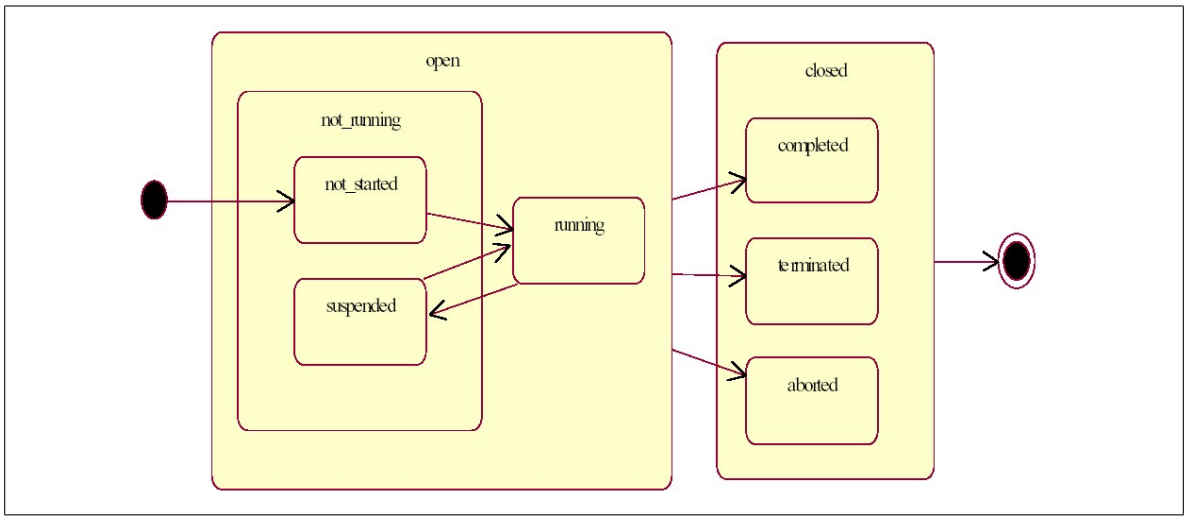


FIG. 3.12 – Les états d'un WfExecutionObject (tiré de [OMG00b])

Les différents états que peuvent avoir les instances d'un *WfExecutionObject* sont représentés à la figure 3.12 et leurs explications sont les suivantes :

Open. Cet état correspond aux instances en cours d'utilisation. Il représente les objects actifs.

Not Running. Les objets sont actifs mais ne sont pas en cours d'exécution.

Not Started. C'est l'état initial d'une instance de `WfExecutionObject` lorsque celle-ci vient d'être créée. Elle est prête à être initialisée et démarrée.

Suspended. L'exécution est temporairement suspendue. Lorsqu'un objet est dans cet état tous les objets qui dépendent de lui sont dans l'état suspendu.

Running. Cet état indique que l'instance est en cours d'exécution.

Closed. Cet état correspond aux instances dont l'exécution est terminée.

Completed. C'est un état final qui indique que l'exécution s'est déroulée et terminée normalement. Cela signifie que tous les objets dépendants sont également dans l'état completed.

Aborted. L'exécution a été annulée avant son achèvement normal. Aucune hypothèse n'est faite concernant l'état des objets dépendant de l'objet annulé.

Terminated. Indique que l'exécution de l'objet a été stoppée avant son achèvement normal. Tous les objets dépendants sont soit dans l'état completed soit dans l'état terminated.

3.2.2.2 Fonctionnement global

Le diagramme de séquence des interactions possibles du WMF de la création à la fin de l'exécution d'un processus est représenté à la figure 3.13. Nous allons maintenant voir plus en détail le fonctionnement.

Création d'une instance de processus. Dans un premier temps, une application implémentant l'interface `WfRequester` demande la création d'une instance de processus au gestionnaire `WfProcessMgr` correspondant par l'appel de la méthode `create_process`. Une fois l'instance de `WfProcess` créée, le gestionnaire renvoie la référence de l'objet au `WfRequester`. Cette nouvelle instance se trouve dans l'état initial *open.not_running.not_started*. Le demandeur de la création peut interroger le gestionnaire pour connaître la signature du processus en appelant les méthodes `get_context_info` (récupération des noms et types des paramètres d'entrée) et `get_result_info` (récupération des noms et types des paramètres de sortie). A partir de ces informations le `WfRequester` peut initialiser convenablement l'instance de processus, en respectant les types des données, via la méthode `set_context`. L'étape suivante consiste à démarrer le processus par l'intermédiaire de la méthode `start`. Il passe alors dans l'état *open.running*, et il va gérer la création des instances de `WfActivity`.

Fonctionnement d'une instance de processus. Lorsqu'un processus est démarré, il commence par déterminer les différentes activités à lancer. Dans notre exemple nous n'avons qu'une seule activité mais il peut bien sûr y en avoir beaucoup plus. Le fonctionnement interne du `WfProcess` n'est pas indiqué (choix des activités, création, ...), la spécification ne s'intéresse qu'aux interfaces des objects. Le processus crée une instance de `WfActivity` représentant le travail à réaliser et l'initialise.

Dans le scénario, `WfActivity` établit une association avec une `WfResource` qui réalisera le travail représenté par l'activité. Cette association est réalisée par l'intermédiaire d'un `WfAssignment`, le mécanisme de sélection des ressources n'est pas sujet à la spécification du WMF.

Le processus démarre ensuite l'activité et l'association potentielle créée précédement se change en une relation réelle car la ressource l'a acceptée. Le statut de l'assignation est alors changé et passe dans l'état accepté.

La ressource réalise le travail, retourne le résultat et invoque l'opération *complete* de l'activité pour prévenir de la fin du travail.

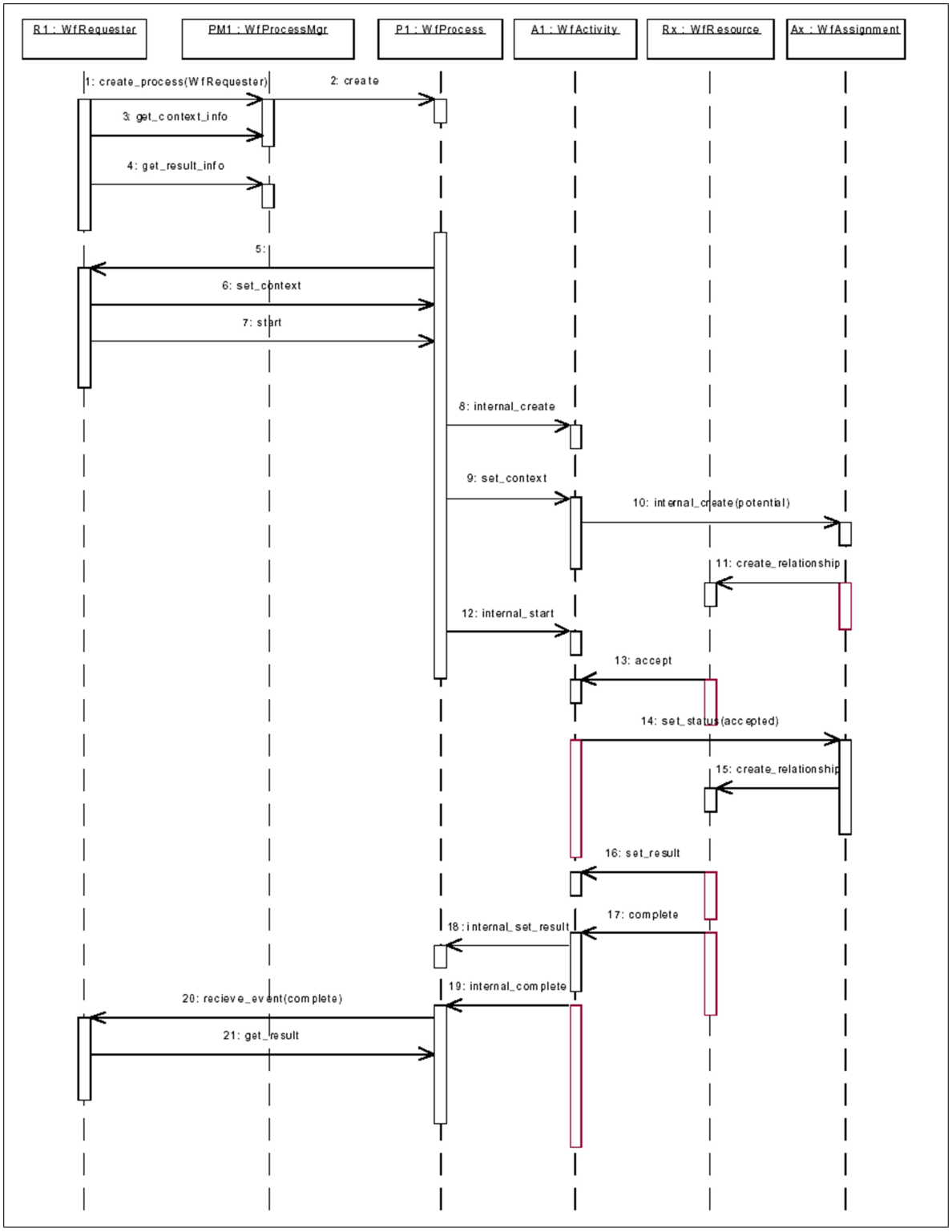


FIG. 3.13 – Fonctionnement global du WMF (tiré de [OMG00b])

L'activité informe le processus de son changement d'état et retourne ses résultats pour un traitement ultérieur par le processus. Il peut utiliser ces informations pour déterminer les prochaines activités à lancer. Dans cet exemple, le processus décide que le travail est terminé et signale sa terminaison à l'initiateur. Ce dernier reçoit la notification de fin et peut récupérer le résultat par l'intermédiaire de la méthode `get_result`.

3.2.2.3 Problèmes liés à cette spécification

Comme toutes les spécifications de l'OMG, celle-ci ne définit que les interfaces sur un moteur de workflows qui peut préexister. À ce titre, la spécification reste floue sur certains aspects tels que le langage de modélisation des processus (aucun langage n'est référencé), la gestion du cycle de vie de certain éléments du workflow (notamment les activités) et le flux de données entre activités et entre activité et processus. Elle reste aussi imprécise sur la gestion de la persistance ce qui peut être gênant dans l'optique où les processus workflows sont par nature de très longues durées pendant lesquelles des problèmes de tous ordres (problèmes matériels, système informatique sous-jacent, ...) peuvent survenir et remettre en cause les instances en cours d'utilisation. La spécification indique seulement l'existence du service de persistence CORBA [OMG00a] pour résoudre ces problèmes. Certaines critiques supplémentaires avaient été émises sur cette proposition alors qu'elle n'était qu'une réponse au RFP[5] [MJ98], comme par exemple le manque de cas d'utilisation UML (*use case*) permettant d'éclaircir le fonctionnement des différents éléments de la spécification et l'absence de référence à des patrons de conception (*design pattern*) pour la mise en œuvre.

Nous devrons répondre à la majorité de ces problèmes dans la mesure où nous avons choisi cette spécification comme cadre d'implantation. Les réponses apportées seront présentées dans les chapitres 4 et 5.

Après avoir défini de manière générale les systèmes de workflows et présenté les principaux standards s'y référant, nous allons maintenant nous intéresser aux différentes fonctionnalités avancées que peuvent posséder de tels systèmes. Nous verrons les deux aspects primordiaux que sont la flexibilité lors de l'exécution et la gestion des exceptions et nous terminerons cette section par un aspect qui nous intéresse particulièrement : la gestion du temps.

3.3 Les fonctionnalités avancées

Dans cette section, nous nous intéresserons aux principaux axes de recherches de ces dernières années dans le domaine des systèmes de workflows. Les principaux problèmes concernant les systèmes de workflows et qui ont été identifiés lors d'un workshop de la NFS [SGJ+96] sont les suivants :

- La majorité des systèmes sont dédiés soit aux processus fortement structurés soit aux processus fortement non structurés alors que la majorité des processus couvre l'espace intermédiaire entre ces deux extrémités (figure 3.14). L'axe horizontal représente le

[5] Request For Proposal

spectre des différents types de processus et l'axe vertical représente la fréquence des occurrences de processus ou des types d'outils.

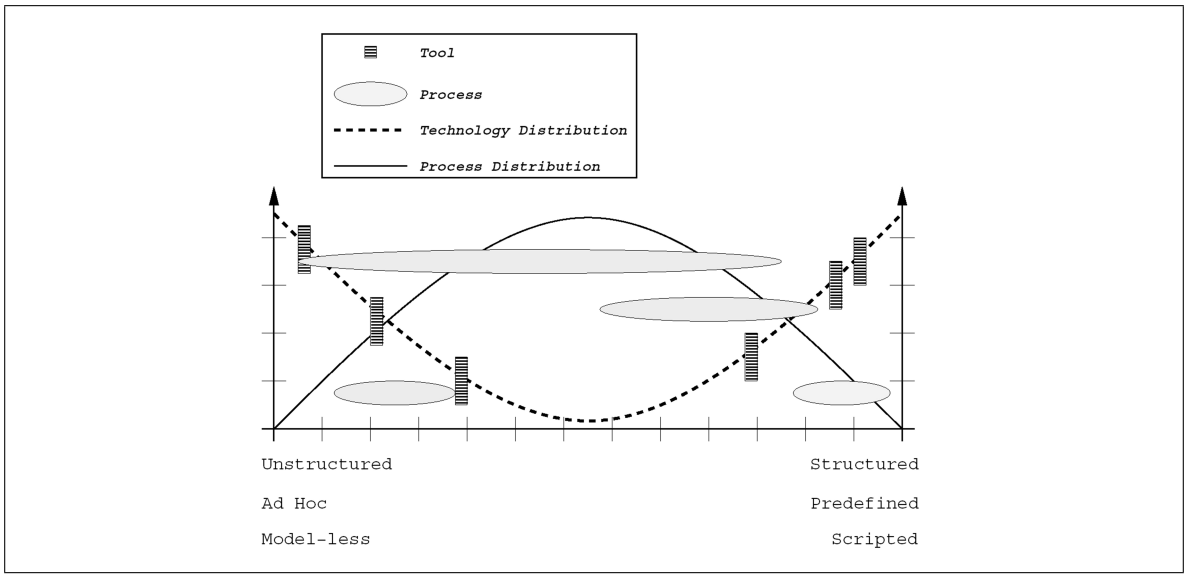


FIG. 3.14 – Support technologique du spectre des processus (tiré de [SGJ+96])

- Quel est le rôle de l'humain dans de tels systèmes ? Dans un premier cas le processus émerge des interactions entre les différents participants et la technologie n'est qu'un médium pour l'interaction. Dans le second cas, les interactions humaines sont complètement gérées par le système qui devient prescripteur du travail à réaliser, laissant peu de place à l'humain. Ce dernier peut n'avoir aucune vision globale du processus dans lequel il intervient.
- La trop grande rigidité des systèmes de workflows ne leurs permet pas d'être adaptés aux différents changements survenant dans leurs environnements et aux souhaits des utilisateurs, ce qui provoque un certain rejet vis-à-vis de ces systèmes. De plus les problèmes et exceptions qui peuvent survenir lors de l'exécution d'un processus ne sont pas obligatoirement gérés.
- Les notions de temps d'exécution et de date limite (*deadline*) ne sont pas suffisament prises en compte notamment pour l'assignation des différentes tâches (risque de surchage de travail pour un utilisateur).

La principale solution pour rémédier à ces difficultés est de rendre les systèmes de workflows flexibles. La flexibilité se décompose en deux catégories : l'adaptabilité des modèles et des plates-formes et la gestion des exceptions. Nous allons maintenant voir plus en détail ces deux domaines.

3.3.1 Adaptabilité

L'adaptation des modèles consiste à fournir un moyen permettant la redéfinition d'un modèle en cours d'exécution pour faire face à une nouvelle situation. Cette propriété nous intéresse particulièrement dans le cadre de la FOAD dans la mesure où l'enseignement est une activité par nature opportuniste et qu'il est souvent nécessaire d'adapter le déroulement d'un module par exemple au niveau des apprenants.

3.3.1.1 Niveaux d'adaptabilité

[HSB98] distingue 4 niveaux différents d'adaptabilité des systèmes de workflows qui vont du plus haut niveau d'abstraction au plus bas :

Adaptation au changement de contexte. Les systèmes de workflows ne fonctionnent généralement pas seuls, ils interagissent avec d'autres systèmes du système d'informations et cela dans un domaine particulier comme par exemple le domaine médical, le domaine financier, *etc.* Des changements contextuels peuvent avoir des répercussions et doivent être pris en compte.

Adaptation des processus. Le modèle peut être manipulé de manière dynamique pour y effectuer des modifications ad hoc. Ce type d'adaptation est nécessaire pour contrôler de manière flexible l'exécution d'un processus. Les deux mécanismes les plus populaires pour l'adaptation des modèles sont une approche par méta-modèle ou une approche par point ouvert (*open-point*).

- Dans la première, le méta-modèle détermine la structure et le type des constituants du modèle de workflow. Un ensemble de primitives est généralement défini, avec lesquelles il est possible de réaliser des opérations de changement du modèle ou des instances de modèle. Les changements peuvent être directement réalisés par les utilisateurs s'ils ont accès à des outils adaptés à leurs compétences et aux concepts qu'ils manipulent. La majeure partie des changements sont d'ordre structurel, ajout, retrait, modification des activités et des liens qui les unissent. Il est également possible de réaliser des changements locaux pour adapter une tâche à un utilisateur particulier lui permettant par exemple de modifier sa tâche pour faire face à un événement imprévu.
- L'approche par point ouvert définit un certain nombre de points dans le modèle où une adaptation peut être réalisée. Ce type d'adaptation est souvent limité à choisir un élément parmi un ensemble prédéfini, à permettre une liaison tardive avec les ressources lors de l'exécution ou à autoriser une modélisation tardive. Modélisation tardive signifie que certains sous-modèles du modèle en cours d'exécution peuvent être définis dynamiquement et immédiatement utilisés. Cependant cette approche peut se révéler insuffisante pour faire face à des situations imprévues [Sad00] [LS97] [KG99] [RD97] [vdA99].

Adaptation des ressources. Ce niveau d'adaptation traite les changements et les réarrangements des ressources manipulées par le système. Les deux principales catégories de changement sont l'adaptation des ressources liées au modèle organisationel (assignations

à d'autres personnes du même rôle par exemple) et l'adaptation relative aux données (changement de type par exemple).

Adaptation de l'infrastructure En réponse à l'évolution des besoins et également aux avancées technologiques, les systèmes logiciels ont besoin d'être rapidement adaptés, ce qui résulte en de nouvelles configurations. L'infrastructure doit donc s'assurer que le système sous-jacent peut évoluer. L'utilisation des dernières avancées dans le domaine des systèmes distribués, du génie logiciel, des intergiciels et des infrastructures à composants logiciels doivent être mis à profit pour la réalisation de systèmes de workflows flexibles [HSB98].

3.3.1.2 Dimensions de la flexiblité

La flexibilité des workflows répond à différents types de problèmes, chacun nécessitant une solution individuelle. Les dimensions de la flexiblités sont les suivants [DGL00] :

Flexibilité du processus. Un modèle de processus définit la séquence et le déroulement des activités à réaliser. Cependant, il y a souvent des situations où des exceptions se produisent. Dans certains domaines il n'est pas possible de complètement définir par avance le modèle de processus.

Flexiblité inter-organisationnelle. Un modèle de processus démarre en se basant sur l'hypothèse que les entités définies sont connues par avance. Cependant dans le cas de partenariat inter-entreprise (organisations virtuelles) le partenariat est susceptible d'évoluer au cours de l'exécution. Il doit être possible d'ajouter de nouveaux fragments dans un processus complet (changement de fournisseur par exemple). Les processus peuvent être distribués géographiquement et peuvent fonctionner sur différentes plates-formes, il existe donc un besoin d'interopérabilité.

Gestion flexible des informations et des connaissances. A l'intérieur du modèle de workflow la circulation des différentes informations nécessaires à l'exécution est prévue. Dans certains cas il est possible d'ajouter des informations en utilisant un principe de dossiers qui circulent entre les différentes activités. Cependant il y a d'autres informations (expériences obtenues en réalisant une activité, informations relatives au client, ...) qui peuvent être intéressantes dans le contexte d'un processus.

Besoins de connaissances flexibles. Il est parfois difficile de prévoir dans le modèle l'intégralité des connaissances et des informations nécessaires. Certaines informations pourront provenir des personnes qui réaliseront les activités.

Diversité des résultats. Les résultats des activités sont souvent documentés de manières différentes. Au-delà des résultats obtenus, la documentation relative à ces résultats (connaissance provenant de l'exécution du processus) est très importante pour l'exécution du processus.

Echange de connaissance entre les différents cas d'utilisation. L'utilisation de processus utilisant de nombreuses informations n'est pas indépendante des autres processus.

Cela requiert une présentation de l'information en fonction du contexte d'utilisation (adaptation au contexte).

Allocation flexible des tâches. Cette assignation des ressources aux activités est en général prédéfinie dans le modèle. Elle est la plupart du temps réalisée via un gestionnaire de rôles. Cependant elle doit être suffisamment flexible pour être redéfinie lors de l'exécution en prenant en compte par exemple certaines méthodes d'allocation (répartition de la charge de travail, choix des tâches pour les personnes détenant le même rôle, informations contextuelles du processus, ...). Le degré d'automatisation peut être divisé en automatique, partiellement automatique et manuel. Manuel implique qu'une personne décide d'affecter les tâches aux autres personnes. Partiellement automatique signifie que le système propose une personne pour réaliser la tâche mais au final un humain choisit. L'allocation doit pouvoir être modifiée au cours du temps pour faire face à certaines situations comme par exemple l'absence imprévisible d'une personne.

3.3.2 Gestion des exceptions

Les systèmes de workflows sont plus particulièrement dédiés aux processus qui suivent un comportement prédictible et répétable. Cependant, des situations exceptionnelles peuvent survenir dans n'importe quel processus et provoquer des dérivations par rapport au comportement normal. Une catégorisation des exceptions et des erreurs pouvant survenir dans un système de workflows fut réalisée par Eder et Liebhart [EL95] et reprise par la suite [Cas98]. Elle distingue quatre types de problèmes : les échecs basiques, les échecs d'applications, les exceptions prévisibles et les exceptions imprévisibles :

Echec basique. Cela correspond à la défaillance d'un des systèmes sous-jacents à la plate-forme de workflows comme un arrêt intempestif du système d'exploitation, une coupure réseau ou l'échec d'une transaction dans une base de données. Le système de workflows doit continuellement sauvegarder son contexte d'exécution afin de redémarrer automatiquement les différents processus dans un état stable avec les données précédant l'incident.

Echec d'applications. Ce type d'erreur provient principalement d'une erreur de programmation d'un module du workflow ou d'une application externe ou d'une violation de contrainte. Le système de workflows va interrompre la tâche fautive et attendre qu'un humain ait résolu le problème. Idéalement, l'administrateur de la plate-forme devrait avoir la possibilité de mettre en place une solution provisoire permettant au processus de continuer sa progression (réaliser la tâche manuellement ou à l'aide d'autres applications, laisser la tâche de côté si cela n'amène pas à des états inconsistants) jusqu'à la correction de l'erreur.

Exception prévisible. La classe la plus importante d'exceptions est celle des exceptions prévisibles, i.e., les situations anormales qui font parties de la sémantique du processus et qui sont connues par avance par le concepteur de processus, comme par exemple l'annulation d'un vol dans un processus de réservation d'un voyage [CP99][HA00]. Comme ces situations peuvent être prévues et font parties de la sémantique du processus il doit

être possible de les modéliser dans la définition du processus. Le workflow et les exceptions prévisibles capturent des aspects différents de la sémantique de l'application. Les workflows représentent le comportement normal alors que les exception prévisibles représentent le comportement occcasionel. Généralement, les exceptions prévisibles ne sont pas très fréquentes mais, quand elles arrivent, elles conduisent à l'exécution d'un processus complètement différent. Les exceptions prévisibles sont souvent asynchrones. Elles sont déclenchées dans un état arbitraire du processus et peuvent intervenir simultanément à une tâche dont l'exécution demande un grand temps d'exécution. Pour cette raison il est souvent difficile de capturer et de représenter les exceptions dans un graphe d'activités qui décrit le comportement d'un processus.

Exception imprévisible. Cette classe d'exception correspond principalement à la nécessité de modifier la structure du processus lors de son exécution pour faire face à un nouveau besoin inattendu. Ce besoin peut être lié à un changement de l'environnement du workflow. Par exemple un changement de loi sur l'obtention de visa oblige une agence de voyage à modifier son processus afin d'obtenir les nouvelles autorisations [Cas98]. Cette catégorie d'exceptions arrive relativement fréquemment car la phase de modélisation est complexe et souvent inexacte ou incomplète et parce qu'un système de workflows est très dépendant de son environnement. Cette catégorie d'exception nous renvoie à la section précédente sur l'adaptabilité des modèles.

On peut également catégoriser les exceptions en fonction de leurs caractéristiques comme par exemple la synchronicité, la portée et l'événément déclencheur [CP99] :

Synchronicité. Les exceptions peuvent être synchrones ou asynchrones par rapport à la progression du workflow. Les exceptions synchrones arrivent en correspondance avec le début ou la fin d'une tâche ou d'un processus, alors que les exceptions asynchrones peuvent apparaître à n'importe quel moment durant l'exécution du processus. Une exception synchrone peut par exemple arriver lors de la modification d'une donnée. La vérification de contraintes sur cette donnée peut se réaliser à la fin de la tâche. Une exception asynchrone peut provenir d'une annulation d'un client. Elle peut survenir à tout moment. Les exceptions synchrones peuvent de plus être caractérisées par leur localisation. Les exceptions localisées ne peuvent arriver que lors de l'exécution d'un petit ensemble de tâches, alors que des exceptions clairsemées (*sparse*) peuvent survenir dans de très nombreuses tâches. Une violation de contrainte sur une donnée est localisée si la donnée n'est manipulée que dans une tâche, clairsemée si elle est manipulée dans de nombreuses tâches.

Portée. Les exceptions peuvent être spécifiques à un processus ou peuvent être inter-processus, i.e. elles sont liées et affectent plusieurs processus. Il existe un lien explicite ou implicite entre les processus. Si un agent réalise différentes tâches pour différents processus, son absence provoquera une exception qui concernera plusieurs processus. De même si une personne réserve un vol et une chambre d'hôtel pour son voyage, l'annulation provoquera une annulation pour chacun des processus(vol et hôtel). Les exceptions peuvent se dérouler sur plusieurs processus pour leurs détections et leurs traitements.

Événement déclencheur. Les exceptions peuvent être levées par différents types d'événements.

Événement temporel (*temporal event*). Ces événements apparaissent à certaines dates (*deadline*) , périodiquement (tous les jours à 13h, par exemple) ou après un intervalle de temps (durée pour réaliser une tâche).

Événement externe (*external event*). Ils sont spécifiés par des applications externes ou des agents intervenant dans le déroulement du processus.

Événement du workflow (*workflow event*). Ils correspondent au début ou la fin d'une tâche (détermination de cycles et de boucles sans fin si une tâche se répète indéfiniment).

Nous allons maintenant passer au dernier aspect qui nous semble important pour un système de workflows, la gestion du temps.

3.3.3 Gestion du temps

L'exécution d'un processus ne se limite pas à l'ordonnancement d'activités en fonction des règles métier. Le système doit également tenir compte de l'aspect temporel [EPPR99] [SME99].

La gestion du temps permet de gérer le planning de l'exécution du workflow dans le temps, de fournir des estimations sur la durée d'exécution des activités, d'éviter la violation des dates limites assignées à des activités et au processus complet et de réagir aux violations des limites lorsqu'elles arrivent. De nombreux processus ont des restrictions temporelles, incluant les durées (généralement maximales, mais parfois minimales) pour les activités et les sous-processus ainsi que des dates limites absolues associées aux activités et aux sous-processus. Dans la plupart des cas, les durées correspondent aux temps d'exécution estimés ou prévus de l'activité. En conséquence la gestion du temps doit être un des éléments du noyau de fonctionnalité fourni par le système de workflows afin de contrôler le cycle de vie d'un processus. Les besoins suivants doivent être satisfaits [EP00] :

1. Les modeleurs de processus ont besoin de moyens pour représenter les aspects temporels pertinents ;
2. Les gestionnaires de processus doivent pouvoir ajuster les temps prévus (extension des limites) en respectant les différentes contraintes temporelles ;
3. Les utilisateurs ont besoin d'informations sur l'urgence des tâches qui leurs sont assignées, afin de gérer leurs plannings ;
4. En cas d'échec lié au temps (limite passée) le système de workflows doir lever une exception afin de prévenir le responsable de l'instance de processus ;
5. Les contrôleurs et les gestionnaires qualité ont besoin d'informations sur le moment où les instances d'activités ont été réalisées ainsi que leurs durées d'exécution.

3.4 Quelques plates-formes de workflows

Nous allons maintenant présenter quelques plates-formes de workflows qui nous semblent intéressantes dans le cadre de nos travaux. Nos principaux critères sont l'architecture utilisée, la modélisation des processus et la gestion de l'adaptabilité.

3.4.1 MetuFlow$_2$

MetuFlow$_2$ [KCD99] est un système de workflows pouvant modifier dynamiquement la structure de ses instances de processus. Une instance est conçue comme un objet qui contient l'ensemble des données nécessaires à son fonctionnement et des informations de contrôle ainsi que l'historique d'exécution.

Ce système utilise un langage de définition de processus de workflow nommé FLOW$_{DL}$ ainsi qu'une représentation graphique FLOW$_{GRAPH}$ et un langage de modification de processus FLOW$_{ML}$. FLOW$_{DL}$ est un langage de spécification structuré par bloc. Les blocs se différencient des tâches et des processus. Ce sont uniquement des activités conceptuelles utilisées pour spécifier l'ordre et les dépendances entre activités. Il décrit les tâches à réaliser d'un processus métier, les dépendances, l'exécution et les données transmises entre les tâches. La spécification des processus est réalisée via un outil graphique (applet Java). FLOW$_{DL}$ contient 8 types de bloc (serial, and_parallel, or_parallel, xor_parallel, for_each, contingency, conditional, iterative). Un processus est défini comme une collection de blocs, de tâches et de sous-processus. Une tâche est la plus simple unité d'exécution. Les processus et les tâches ont des paramètres d'entrée/sortie correspondant aux donnés servant à communiquer avec les autres processus ou tâches. Le terme activité est utilisé pour se référer à un bloc, une tâche ou un processus. Il y a cinq types de tâches (Transactional, non_transactionnal with checkpoint, non_transactionnal, user et 2PC_transactional) avec les attributs suivants : critical, non_vital, critical_non_vital.

3.4.1.1 Architecture

L'architecture globale de Metuflow$_2$ (figure 3.15) est constituée de quatre composants principaux :

Component Server Repository. Les composants du système sont implémentés en tant qu'objets CORBA invoqués par des applets Java. Le Component Server Repository contient ces applets.

Workflow Process Definitions Library. Ce composant gére les modèles de processus, défini les rôles organisationnels et l'assignation rôle-participant. Ce composant gère un historique des versions. La version par défaut est la dernière en date.

History manager. Ce composant gère l'historique des processus terminés. L'historique des processus en cours est stocké à l'intérieur même de l'objet processus [KAD98].

Workflow Domain Manager Ce composant est un serveur web donnant accès aux différentes applets. Il sert de portail à l'application (authentification, fourniture d'applets en fonction du profil de l'utilisateur, ...).

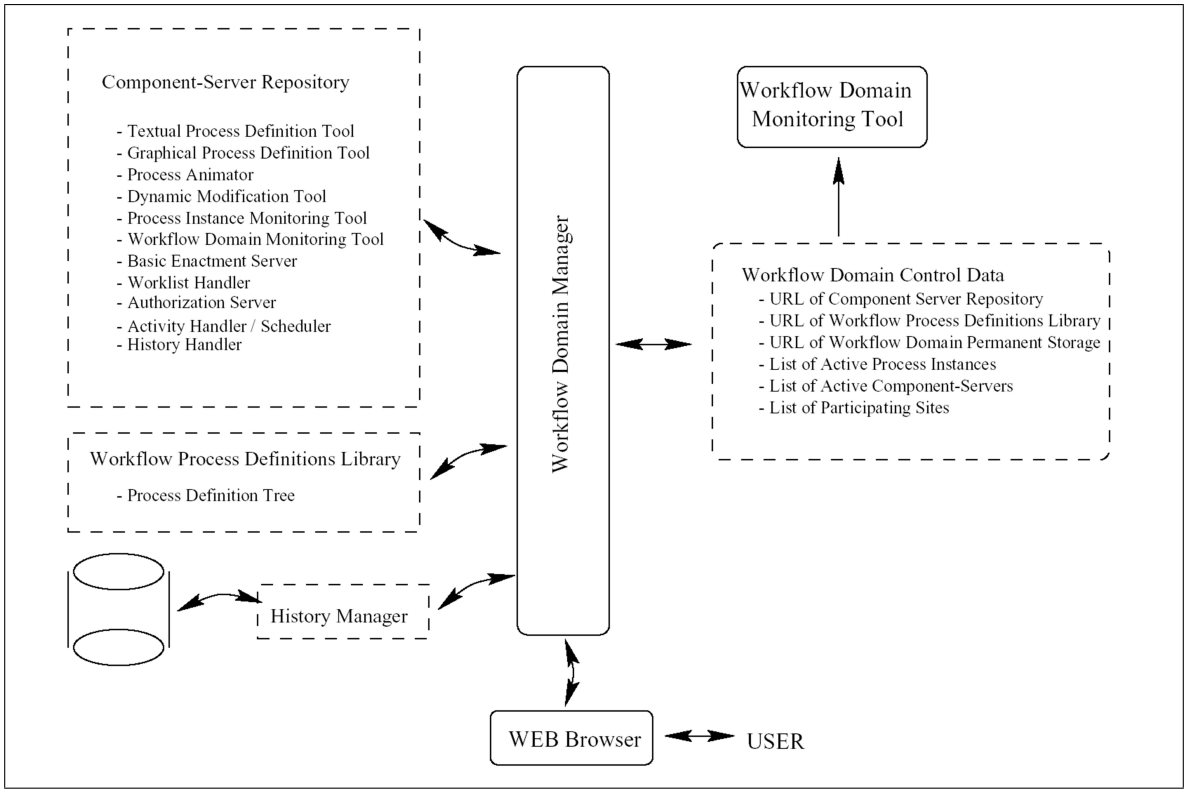


FIG. 3.15 – Architecture de Metuflow$_2$ (tirée de [KCD99])

3.4.1.2 Modification du processus

Les modifications sont exprimées dans un langage nommé FLOW$_{ML}$. Il permet :
- la description d'une action de modification (`add`, `delete`, `modif`) sur un élément du modèle (`process`, `task`, `block`) ;
- la désignation de l'endroit où la modification est appliquée ;
- la spécification d'un domaine de modification, i.e., les instances affectées par le changement ;
- le choix du type de modification, i.e., modification permanente (affectant les nouvelles instances) ou temporaire (localisé aux instances en cours d'utilisation).

L'une des particularités de cette plate-forme est de disposer d'un type particulier d'activité nommé *Dynamic Workflow Special Activity* (DWSA). Elle permet de démarrer un workflow sans spécifier complètement le modèle. Il ne contient que cette activité spéciale. Cela permet de construire dynamiquement son processus en spécifiant les tâches et en les réalisant immédiatement.

3.4.1.3 Conclusion

Les principaux avantages que nous avons retenus de cette plate-forme sont :

- Une architecture modulaire avec une décomposition claire des différentes fonctionnalités ;
- La capacité de réaliser les modifications en utilisant un langage aproprié. Cela formalise les changements possibles ;
- La capacité de créer complètement de manière dynamique une nouvelle instance de processus.

3.4.2 Flex-eL

Le système Flex-eL [MO00] (Flexible e-Learning) est conçu par le Distributed Systems Technology Center (DSTC) [DST03] et par l'université du Queensland en Australie. Flex-eL est une plate-forme d'éducation à distance basée sur un moteur de workflows flexible. Les créateurs sont partis du principe que les systèmes d'enseignement à distance existants se focalisaient plus sur les technologies que sur le processus d'apprentissage. L'objectif n'est pas de recréer ce que l'on trouve de manière traditionnelle dans l'enseignement mais d'inventer un nouvel environnement d'apprentissage.

Ce projet, débuté en mars 2000, est actuellement testé à l'université du Queensland dans le cadre d'un cours du Master of Information Technology. Les premiers résultats semblent positifs notamment du point de vue du taux de désaffection.

L'approche suivie consiste à créer un apprentissage centré sur l'utilisateur. Le workflow fournirait quant à lui la bonne tâche au bon moment. De plus il fournit une bonne intégration des nouvelles ressources et des nouveaux outils lorsqu'ils deviennent disponibles dans le futur.

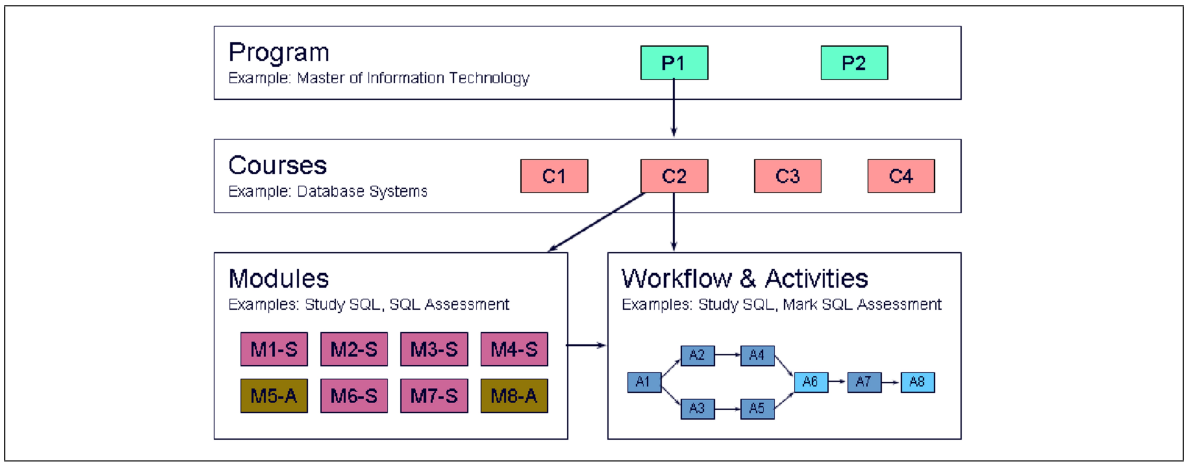


FIG. 3.16 – Modélisation des cours dans Flex-eL (tiré de [LHSO02])

La figure 3.16 montre la structure d'un cours de Flex-eL. Elle est similaire à la structure d'un cours traditionnel. Au plus haut niveau on trouve la formation. Chaque formation est composée d'une série de cours correspondants à des domaines particuliers qui sont constitués d'un ensemble de modules. Dans une approche traditionnelle les modules d'un cours sont réalisés de manière séquentielle. Dans Flex-eL les cours peuvent être conçus de telle manière

que les modules puissent être réalisés parallèlement ou séquentiellement. C'est une modalité d'apprentissage plus flexible et les étudiants peuvent décider de leurs propres cheminements, le module est conçu comme un processus de workflow. Dans l'environnement traditionnel d'apprentissage les modules ont des durées et des dates limites fixes, ce qui implique de nombreuses contraintes temporelles.

3.4.2.1 Architecture

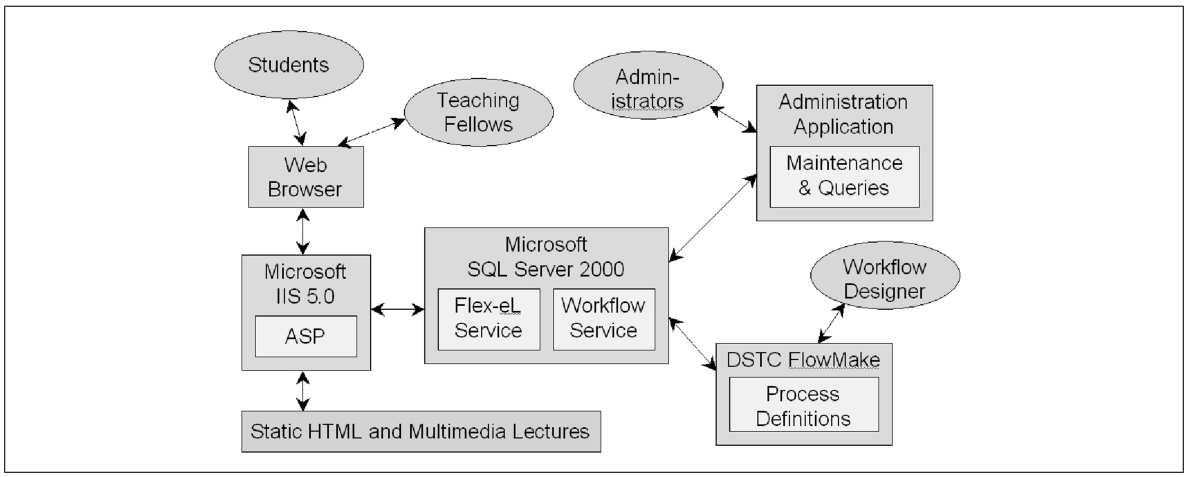


FIG. 3.17 – Architecture de Flex-eL (tiré de [LHSO02])

L'épine dorsale de Flex-eL est l'utilisation des technologies workflows et contient d'autres outils et technologies pour fournir un environnement de e-learning complet. Les différents constituants de l'architecture sont présentés à la figure 3.17. Un outil de modélisation de processus workflows développé par le DSTC, FlowMake, est utilisé pour la conception des processus d'apprentissage. L'accès du système par les étudiants et les enseignants à lieu par l'intermédiaire de pages web dynamiques. Un ensemble d'outils permet de gérer l'inscription et le suivi des étudiants et d'administrer les modèles et les différentes instances de processus.

3.4.2.2 Fonctionnement

Lorsqu'un étudiant s'inscrit dans un cours, une nouvelle instance du modèle de cours est créé et assigné à l'étudiant. C'est une des méthodes de Flex-eL permettant d'offrir un cheminement dynamique adapté et flexible pour chaque étudiant. Chaque étudiant peut apprendre à son propre rythme sans s'inquiéter d'éventuel dates limites. En relachant les contraintes temporelles, la flexibilité de la gestion temporelle personnelle est réalisée. Cela permet également à certains étudiants de terminer leurs cursus plus rapidement ou de répartir leurs charges de travail en tenant compte de leurs contraintes extérieures. Durant les phases d'apprentissage, la progression de chaque étudiant est capturé par le système de workflows. Ainsi les étudiants ont la possibilité d'obtenir des informations sur les autres étudiants réalisant

la même activité. Cela encourage la collaboration en identifiant mieux les éventuels sujets d'intérêts que par la simple mise à disposition d'outils de discussion synchrone ou asynchrone. Une des autres caractéristiques importantes de Flex-eL est la surveillance de la progression à travers la visualisation du workflow. Cela fourni les informations nécessaires pour comprendre les interactions entre les tâches workflows et les processus de décision. Les étudiants peuvent ainsi visualiser le chemin parcouru et également le chemin qui reste à faire (cela leurs permet de mieux planifier leur travail par exemple).

3.4.2.3 Conclusion

Cette plate-forme répond bien au contexte de la formation continue. Néanmoins, elle ne répond pas complètement à notre problématique aussi bien sur le contexte d'utilisation que sur l'approche technologique. Nous voulons un système utilisable à la fois dans un contexte de formation continue et dans un contexte de formation traditionnelle. Nous souhaitons spécifier de véritables tâches collaboratives où les étudiants travaillent ensemble.

3.4.3 Micro-Workflow

La structure cadre Micro-Workflow fut conçue par Dragos-Anton Manolescu au cours de son doctorat [Man00]. Il s'agit d'une architecture de workflows flexible et extensible à destination des développeurs d'applications. Le constat à l'origine de ces travaux est qu'il existe un grand nombre de systèmes de workflows mais qu'ils sont basés sur une architecture ne permettant pas aux développeurs de les réutiliser dans un cadre de développement orienté objet. Les technologies objets sont souvent utilisées pour l'implémentation mais elles ne transparaissent pas à l'extérieur de ces systèmes monolithiques. Il n'est pas possible de réutiliser ces objets ni de les modifier ou les étendre afin d'ajuster ou d'ajouter des fonctionnalités pour répondre à des besoins spécifiques. De plus il est difficile d'intégrer ces systèmes dans des applications nécessitants des services de type gestion de processus et d'activités.

3.4.3.1 Architecture

La philosophie derrière l'architecture réside dans l'utilisation des concepts de la programmation orientée objets et des patrons de conception. Elle offre des possibilités d'extension par des mécanismes de boîtes blanches et de boîtes noires. Afin de résoudre le problème de "lourdeur", Micro-workflow s'appuie sur un noyau minimaliste ne comportant que les fonctions de bases liées à la définition et à l'exécution de workflows. Le modèle de processus utilisé est basé sur un réseau d'activités. Lors de l'exécution toutes les entités sont représentées par des objets. Le noyau est composé de trois éléments : un composant d'exécution, un composant de synchronisation et un composant de processus. Le composant d'exécution assure le déroulement du workflow selon son modèle.

L'architecture complète de micro-workflow est visible à la figure 3.19. En plus du noyau, et afin de montrer les caractéristiques de flexibilité et d'extensibilité de la plate-forme sept composants furent développés :

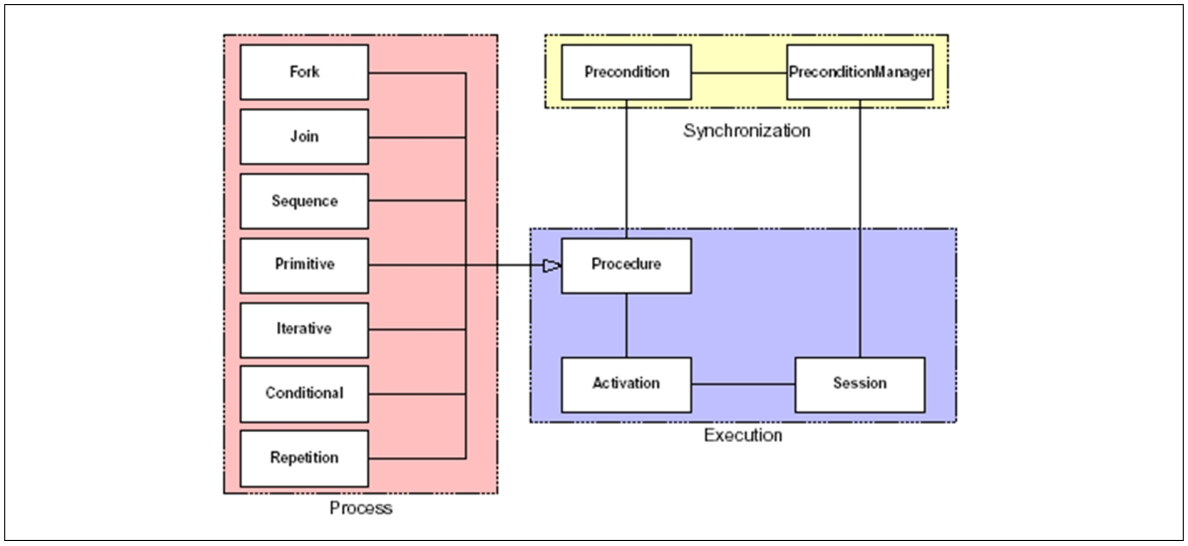


FIG. 3.18 – Noyau de Micro-workflow (tiré de [Man03])

Un composant d'historique. Ce composant extrait les événements du composant d'exécution et les sauvegarde afin d'être utilisés ultérieurement pour analyser le comportement des instances.

Composant de surveillance. Ce composant permet de vérifier le fonctionnement des différentes instances en cours d'exécution.

Composant de persistance. Ce composant se charge de la sauvegarde des états et des données des différents objets.

Composant de gestion de listes de tâches. Ce composant donne accès aux tâches des utilisateurs.

Composant d'intervention manuelle. Ce composant est utilisé pour modifier l'instance en cours d'exécution. Il offre ainsi des fonctionnalités d'adaptabilité aux utilisateurs.

Composant de fédération de workflows. Ce composant [MJ00] permet la répartition de l'exécution des processus.

3.4.3.2 Conclusion

Cette approche par micronoyau se révéle intéressante. Elle permet de bien isoler le cœur de la plate-forme et permet l'ajout de composants spécifiques à la demande afin d'adapter la plate-forme à des besoins particuliers. Néanmoins, micro-workflow s'adresse uniquement aux développeur d'applications. Il est presque impossible pour un non-informaticien de créer un nouveau modèle de processus.

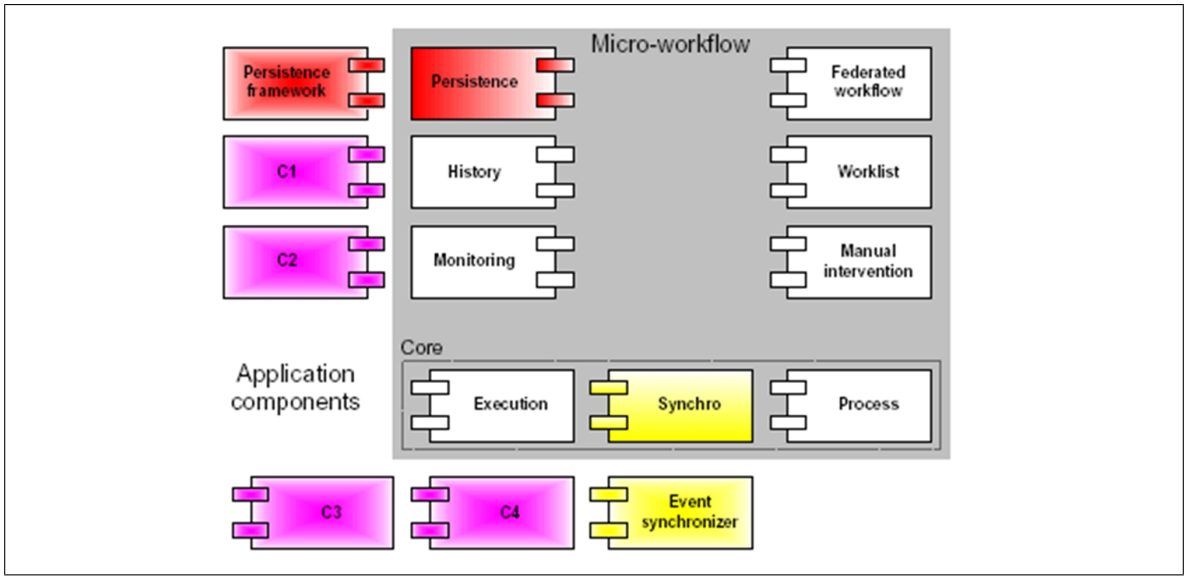


FIG. 3.19 – Architecture de Micro-workflow (tiré de [Man03])

3.4.4 Conclusion

Suite à la présentation de ces quelques systèmes de workflows, nous pouvons retenir les points suivants pour notre conception :

Modularité du système. Les plates-formes présentées sont toutes consistuées de composants réalisant chacun une tâche bien précise. Ce découpage évite la construction d'un système monolithique difficile à maintenir et à faire évoluer. L'approche par micro-noyau nous semble la plus prometteuse car elle implique un minimum de contraintes pour les composants additionels. Cette modularité peut également être augmentée avec le principe de boîte blanche pour la création des composants.

Spécification des modifications. La formalisation des changements et leur expression par un langage permet une manipulation plus facile des instances. Cela permet de garder plus facilement une trace de tous les changements qui ont été réalisés.

Il faut également remarquer qu'il existe peu de gestion du temps dans les plates-formes que nous avons présentées.

3.5 Conclusion du chapitre

Au cours de ce chapitre, nous avons défini les systèmes de workflows en présentant notamment leurs architectures générales et les différents standards existants. Nous avons présenté les différents éléments de standardisation des systèmes de workflows. De plus en plus de produits commerciaux respectent ces normes. Nous nous servirons de ces standards comme cadre de

conception, cela assurera une meilleure pérennité à notre système et permettra de profiter de l'existant (outils de modélisation, ...).

Enfin nous avons terminé ce chapitre par l'étude des fonctionnalités importantes dans notre cadre de travail : la flexibilité, la gestion des exceptions et la gestion du temps. L'étude de quelques moteurs de workflows existants nous a permis de montrer quelques exemples de la manière dont ces propriétés peuvent être prises en charge. La participation des utilisateurs étant prépondérante dans le cadre de la formation, il est certain que notre moteur de workflows doit offrir un haut niveau de flexibilité dans le déroulement des processus. Il devra notamment permettre une modification des activités et des processus aussi bien au niveau individuel que pour des groupes, laisser une liberté dans l'utilisation des outils, effectuer une allocation souple des tâches par exemple dans le cadre des activités coopératives. Enfin, il doit fournir les moyens de gérer les exceptions liées aux aspects temporels qui sont importants dans le cadre de la formation. Pour offrir cette flexibilité, nous avons choisi d'utiliser simultanément une approche par méta-modèle et une approche par point ouvert afin de profiter au mieux des avantages de ces deux solutions. Nous présenterons cela plus en détail dans les chapitres 4 et 5.

Deuxième partie

Etude et conception d'un système d'exécution de scénarios pédagogiques

Chapitre 4

Conception de COW

> UNIX was not designed to stop you from doing stupid things, because that would also stop you from doing clever things.
>
> **Doug Gwyn**

Dans la première partie de ce document nous avons exposé les langages de modélisation pédagogique et dressé un panorama des systèmes de workflows aussi bien concernant les standards que les axes de recherches actuels. Nous avons également présenté quelques réalisations afin d'étudier les méthodes et les moyens mis en œuvre pour répondre aux besoins de flexibilité et de malléabilité lors de l'exécution de processus. Nous allons nous intéresser dans ce chapitre à l'architecture globale de notre proposition. Nous montrerons dans un premier temps (section 4.1) nos objectifs en terme de fonctionnalités et de réalisations en nous basant sur notre étude de la première partie. Nous traiterons ensuite la partie modélisation et méta-modélisation de notre plate-forme (section 4.2), puis nous verrons les moyens mis en œuvre pour réaliser son implantation. La section 4.3 présentera les différentes technologies et concepts utilisables pour atteindre nos objectifs. La section 4.4 décrira la mise en œuvre de notre architecture. Enfin la section 4.5 concluera ce chapitre.

4.1 Objectifs

La réalisation d'un moteur de workflows flexible est devenu un sujet courant depuis bientôt une décennie. Il suffit pour s'en rendre compte de voir le nombre de systèmes industriels ou provenant des laboratoires qui ont été produits récemment. Toutefois, peu ont été réalisés dans le cadre d'environnements standardisés tels que ceux du WfMC et du WMF de l'OMG et encore moins furent conçus pour le monde de l'enseignement à distance. La première partie de ce document nous a permis de constater que les langages de modélisation pédagogique les plus généralistes comme par exemple IMS-LD reposent sur le concept central d'activité.

De l'autre côté, les différents systèmes de workflows mettent en avant ou s'intéressent plus à la notion de processus. Les premiers se focalisent sur la modélisation alors que les seconds s'intéressent principalement à l'exécution. Les EML ont en effet tendance à disparaître lors de l'exécution, ils ne sont réellement exploités que dans les phases de conception et pour les échanges entre plates-formes. Il serait pourtant intéressant de pouvoir les retrouver tout au long du cycle d'utilisation.

Notre proposition COW[6] vise à produire un pont entre les EML centrés sur le concept d'activités et les systèmes de workflows afin de permettre l'exécution de scénarios pédagogiques en tenant compte des caractéristiques issues des deux domaines. Pour atteindre ce but nous avons trois objectifs principaux :

- Le premier objectif est de concevoir une plate-forme **centrée sur l'utilisateur** [BDT01]. Ce dernier intervient tout au long du cycle de vie du système. Il participe à la création des différents modèles (orientés pédagogie ou orientés workflow) et intervient dans la configuration initiale de la plate-forme en paramétrant le comportement de cette dernière en fonction du contexte courant d'utilisation. Il modifiera ce comportement afin qu'il reflète les changements de contextes. Il est co-constructeur de son environnement [Bou00].
- Le deuxième objectif, qui est lié au premier, est de repositionner les **modèles au cœur du système**. Ils doivent être constamment disponibles et manipulables afin de conserver une vue orientée métier de la plate-forme. Cela permettra aux utilisateurs d'adapter plus facilement le modèle à leurs besoins.
- Le troisième objectif, lié aux deux précédents est d'ordre technique. La plate-forme doit mettre à disposition un ensemble de mécanismes de **flexiblité** permettant la manipulation de tous les éléments constituants notre système et leur modification de manière dynamique. Ces changements peuvent intervenir à différents niveaux d'abstraction. Le plus haut niveau d'abstraction concerne le langage de modélisation pédagogique. Nous devons offrir la possibilité de redéfinir tout ou partie d'un scénario pédagogique. Le plus bas niveau d'abstraction concerne l'adaptation du comportement des différents composants logiciels qui constituent notre plate-forme.

Nous allons maintenant voir un peu plus en détail le cadre que nous nous sommes fixé pour atteindre nos objectifs :

L'intégration. Notre travail se focalise sur une plate-forme d'exécution de scénarios pédagogiques. Nous ne cherchons pas à recréer complètement un nouvel LMS[7] mais plutôt à fournir un nouveau type de service qui pourra être intégré dans les LMS existants. Cela signifie notamment que notre système doit pouvoir fonctionner dans un environnement

[6] L'acronyme COW signifie Cooperative Open Workflows. En langue anglaise, COW ne se traduit pas uniquement par ce charmant animal ruminant que nous trouvons dans nos vertes campagnes. Il a également une autre signification qui est « intimider », ce qui résume fort bien la situation de toute personne face à un nouveau système informatique, ou d'un doctorant face à la page blanche ;-)

[7] **L**earning **M**anagement **S**ystem

hétérogène aussi bien du point de vue des systèmes d'exploitation que des langages de programmation utilisés.

Le respect des standards. Une des clés pour l'intégration et l'utilisation de notre plate-forme réside dans le respect des standards des différents domaines que nous abordons. Cela signifie notamment le support de langages existants pour la description des scénarios pédagogiques. Concernant le système d'exécution nous devrons nous conformer le plus possible aux directives du WfMC et de l'OMG.

La réutilisation et la généralisation. Une de nos préoccupation est la réutilisation de l'existant. Cette caractéristique se situe à deux niveaux différents. Le niveau le plus élevé concerne la réutilisation des modèles existants. Cette contrainte est notamment un des fondements des EML que nous avons présentés dans le chapitre 2. Le deuxième niveau de réutilisation concerne le moteur, ses composants internes et les applications externes. Bien que notre domaine d'application soit plus particulièrement l'éducation à distance, notre démarche devra être la plus généraliste possible et dans tous les cas indépendantes d'un EML particulier. Notre système sera ainsi plus à même de répondre aux évolutions futures. Nous ne souhaitons pas concevoir d'applications externes spécifiques (outils de QCM, gestionnaire de fichier, ...) mais nous voulons réutiliser le maximum d'applications existantes. Cette approche généraliste nous amène directement au point suivant, l'extensibilité.

L'extensibilité. Toute activité humaine est par définition imprévisible. Il est donc utopique de croire que nous pouvons réaliser un système répondant à tous les besoins présents et surtout futurs. Afin d'adopter une démarche pérenne, nous devons être en mesure de pouvoir à la fois modifier le comportement de notre système, et également construire une architecture permettant l'ajout de fonctionnalités non prévues à l'heure actuelle.

Le fonctionnement synchrone et asynchrone. Une des solutions permettant l'ajout de fonctionnalités et l'intégration de nouveaux composants réside dans une communication anonyme. Le système rend compte de son exécution en envoyant une série d'événements représentant les différents changements de données, d'états, etc.

Le support de la charge. Notre système à pour vocation d'être utilisé dans des systèmes existants et plus particulièrement dans « le Campus Virtuel ». Nous devons être en mesure de supporter simultanément un très grand nombre d'utilisateurs et d'instances de modèles.

Afin de réaliser nos objectifs dans le respect de ce cadre de travail, nous avons travaillé selon deux axes, une approche par méta-modèle et une approche par composants logiciels et implémentation ouverte, qui sont présentés dans la suite de ce chapitre.

4.2 Approche par méta-modèle

Comme nous l'avons indiqué dans le chapitre 3, l'approche par méta-modèle est une des solutions envisageables pour réaliser l'adaptation des modèles en cours d'exécution. Nous

allons voir plus en détail dans cette section pourquoi cette démarche nous semble intéressante et nous présenterons sa mise en œuvre dans notre système.

4.2.1 Pourquoi une approche par méta-modèle ?

La première question que nous pouvons nous poser est pourquoi baser notre démarche de conception sur une approche par méta-modèle ? La première raison que nous pouvons invoquer pour l'utilisation de cette démarche est qu'elle a été clairement identifiée comme étant un des moyens possibles pour la flexiblité des moteurs de workflows [HSB98, LS97, Kra00]. Au delà des systèmes de workflows, cette approche a été utilisée dans différents projets relatifs au TCAO [Bou00]. La deuxième raison qui nous pousse vers l'utilisation de méta-modèles se situe dans le courant actuel au sein de la communauté génie logiciel d'augmenter le niveau d'abstraction en se focalisant plus sur les modèles que sur leurs implémentations. Cette démarche est au cœur du *Model Driven Architecture* (MDA) de l'OMG [Arc03b] [BG01].

4.2.2 Pourquoi un nouveau méta-modèle ?

Maintenant que nous avons présenté les différentes raisons qui nous poussent à utiliser une approche par méta-modèle, la question que nous pouvons nous poser est pourquoi recréer un nouveau méta-modèle alors qu'il en existe déjà tant [Kra00, LS97] et qu'il en existe même un défini par le WfMC [Wor02] ? Une première explication à cette prolifération peut se retrouver dans la définition du terme méta-modèle telle que donnée par [Met03] :

> *« [metamodels] are an attempt at describing the world around us for a particular purpose. »*

Un méta-modèle décrit le monde dans un but particulier. Or chaque système de workflows apporte une solution à un problème particulier. Il n'existe pas de système suffisamment généraliste pour répondre à l'ensemble des besoins.

Alors pourquoi proposer un méta-modèle standard ? Ce méta-modèle fédère les différents concepts communs à la majorité des systèmes de workflows, mais son objectif est de permettre l'échange de modèles. Il n'est pas destiné à être utilisé pour l'exécution des modèles. Il sert uniquement de passerelle (plus précisément de pivot) pour passer d'un méta-modèle à l'autre. Certains systèmes disposent d'un traducteur pour réaliser l'export de leurs modèles en WPDL/XPDL [Tom99].

Nous pouvons également nous demander pourquoi ne réutiliserions-nous pas le méta-modèle présenté par le CEN/ISSS que nous avons vu au chapitre 2 [RvRK+02]. Il existe au moins deux raisons. Nous souhaitons adopter une démarche générique pour la conception de notre moteur de workflows, i.e., nous ne voulons pas qu'il soit exclusivement dédié à un domaine particulier. La deuxième raison est temporelle, le méta-modèle du CEN/ISSS a été publié bien après le début de nos travaux.

4.2.3 Le méta-modèle de COW

Nous allons maintenant voir le méta-modèle que nous avons défini pour notre plate-forme.

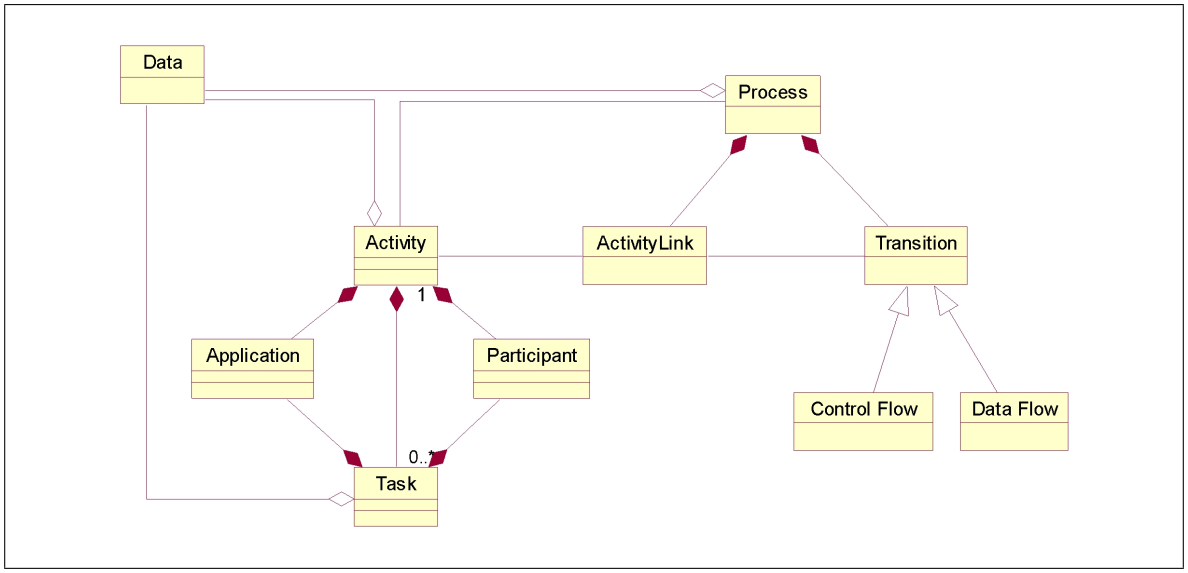


FIG. 4.1 – Méta-modèle de COW

Les deux concepts fondamentaux que nous devons prendre en compte sont la notion de processus et la notion d'activité. Le premier est principalement utilisé dans les systèmes de workflows alors que le second est considéré comme l'élément central pour la modélisation pédagogique.

Les modèles manipulés par COW sont exprimés à partir du métamodèle présenté à la figure 4.1. Nous avons construits notre métamodèle en nous basant à l'origine sur celui de XPDL, mais nous l'avons modifié pour répondre à nos besoins spécifiques. Les différents éléménts de ce méta-modèle sont définis dans la suite.

4.2.3.1 Les processus

Pour permettre une plus grande flexibilité, nous avons séparé les concepts de processus et d'activité. Les activités ne sont plus aggrégées au sein d'un processus. Elles peuvent exister de manière indépendante. Cela permet une plus grande réutilisabilité des modèles d'activités et d'avoir une double vue, une vue orientée activité et une vue orientée processus (coordination implicite/faible vs coordination explicite/forte). Un processus est essentiellement composé de deux entités, *Transition* et *ActivityLink*.

L'entité ActivityLink représente une activité dans le modèle de processus. Elle se compose d'un identifiant unique au sein du processus, d'un identifiant de modèle d'activité et des conditions d'entrée et de sortie spécifiques à l'activité dans le modèle de processus (pré- et postcondition). Ces pré- et postcondition permettent d'exprimer la sémantique lorsque plusieurs transitions arrivent dans un activityLink (Join) ou sortent (split).

L'entité Transition se décompose en deux entités, *control flow* et *data flow*. Control flow (flux de contrôle) détermine les liens de coordination entre les différents ActivityLink, i.e. détermine les chemins possibles à parcourir. Des conditions peuvent être associées à ces tran-

sitions. Data Flow représente le flux de donnés, i.e., le chemin de circulation des données entre les différents ActivityLink. De même que pour le flot de contrôle, nous pouvons y ajouter des conditions.

D'un point de vue implantation, afin de ne pas imposer de langage d'expression des conditions, nous avons utilisé le patron stratégie [GHJV95] pour charger dynamiquement l'interpréteur de langage.

4.2.3.2 Les activités

L'activité est l'élément central dans les modèlisations pédagogiques [IMS03b]. Elle va déterminer le travail à réaliser. Nous distinguons trois types d'activités.

- Le premier est une activité faisant appel à un processus. Cela nous permet d'avoir la notion de sous-processus et ainsi de réaliser la composition de processus et favoriser la réutilisation de modèles existants. Nous avons ajouté un attribut nommé `Group` nous permettant de spécifier si le processus doit être réalisé individuellement par chaque membre du groupe. Ce point sera explicité dans l'exemple de la section 4.2.4.2.
- Le deuxième type est une activité dite de routage (comme dans XPDL). Cette activité est vide, i.e. elle ne contient aucune définition de travail à réaliser. Comme son nom l'indique elle sert de routeur et/ou de synchroniseur entre différentes transitions entrantes/sortantes. Elle permet de simplifier la description du flux de contrôle.
- Le troisième type est une activité d'implémentation, i.e. elle représente un travail à effectuer. Dans ce cadre, l'activité représente un contexte d'exécution identique pour toutes les tâches. Si une activité ne représente qu'une seule tâche à réaliser, l'élément Task devient alors optionnel car il peut exister de manière implicite. Une tâche est associée à des participants et à des applications.

Dans le méta-modèle du XPDL, on ne voit apparaître que le concept d'activité. Lors de l'instantiation, ce concept est mis en œuvre par l'intermédiaire de workitems qui représentent donc les instances d'un modèle d'activité. Une instantiation d'activité peut conduire à la création de plusieurs workitems lorsqu'une activité est réalisée par plusieurs personnes. Un exemple évident dans le domaine de l'éducation à distance est une activité d'évaluation. Chaque étudiant doit réaliser le questionnaire. Cependant, nous avons choisi d'intégrer le concept de workitem au niveau du méta-modèle afin d'offrir une plus grande liberté de modélisation. Une activité peut contenir de zéro à plusieurs tâches. Lorsqu'une activité ne contient pas de tâche, nous nous retrouvons dans le cas du XDPL standard. Cependant, si l'on considère une activité comme un contexte particulier pour la réalisation d'un objectif, nous pouvons définir différentes tâches pour atteindre l'objectif. Pourquoi dans ce cas là ne pas se servir d'un processus ? Tout simplement car il n'existe pas de contrôle explicite de la coordination entre les tâches. Par exemple, dans le cas d'une activité d'apprentissage d'un cours, les différentes tâches peuvent être la lecture du document 1 suivi du document 2 suivi d'un test d'auto-évaluation. Ici l'ordre d'exécution n'est pas imposé mais conseillé. Un étudiant peut directement commencer par l'auto-évaluation.

Notre modèle d'activité s'inspire de celui de la Théorie de l'Activité [Bou00] [Nar95]. Nous avons repris la notion de division du travail et les notions activité/action/opération.

Dans notre approche le concept d'action est considéré comme une tâche à réaliser. Elle est représentée par l'entité WfTask.

Au final, nous utilisons un découpage en trois niveaux (processus, activité, tâche).

4.2.3.3 Les participants et les applications

L'entité participant reprend la même définition que celle du XPDL. L'entité application est basée sur XPDL (pour le langage de description) mais la sémantique est différente. Nous ne souhaitons pas, pour des raisons de flexibilité, que l'activité ou la tâche soit associée ad eternam à une application particulière. Ainsi, l'application ne représente qu'un identifiant avec une catégorie (cela pose le problème de la catégorisation des applications et ensuite des méthodes pour les retrouver de manière dynamique en utilisant par exemple un service de courtage).

4.2.3.4 Extension du métamodèle

Notre métamodèle répond à nos préoccupations actuelles et dans un contexte donné. Afin de permettre d'ajouter des éléments aux différentes entités de notre métamodèle, nous avons repris le concept d'`ExtendedAttributes` du XDPL. Il est possible d'ajouter à chaque entité des éléments `ExtendedAttribute` consistué d'un nom et d'une valeur. Ils permettent de prendre en compte des éléments spécifiques d'un métamodèle. Ils pourront être utile pour le support des EML où certaines caractéristiques n'entrant pas en compte directement pour le fonctionnement du moteur (notion d'objectifs par exemple) doivent tout de même être sauvegardé pour fournir un ensemble d'informations supplémentaires aux utilisateurs. Ceci sera explicité dans le chapitre 6.

4.2.4 Modèles et instantiation

Dans cette subsection nous nous intéresserons au stockage des modèles et à leurs instantiation.

4.2.4.1 Séparation modèle et paramétres

Un des éléments de la réutilisabilité des modèles est leur parfaite indépendance vis à vis des technologies et des plates-formes d'exécution. Dans ce cadre là, un modèle ne peut pas contenir de référence directe à un outil particulier. L'inaccessbilité temporaire ou définitive de cet artefact remettrait en cause le modèle. Pour éviter cela, les modèles ne comportent que des identifiants représentant les outils ou les rôles. Les liens entre ces identifiants et les outils ou personnes réels sont spécifiés dans un modèle d'instance. Pour chaque identifiant, il a entre 0 et n éléments associés. Il peut au premier abord être surprenant de trouver un nombre égal à 0. Si nous prenons le cas d'un outil, ce nombre peut résulter d'un oubli, ou peut être volontaire. L'enseignant/tuteur/ingénieur pédagogique peut dans un contexte particulier laisser les différents acteurs réaliser la tâche de manière complétement manuelle ou leurs laisser le choix du ou des outils à utiliser. Afin d'autoriser ce comportement, notre moteur

devra permettre l'ajout et le retrait dynamique d'outils pour un acteur ou un groupe d'acteurs et ceci pour une ou plusieurs tâches. Chaque modification sera ensuite sauvegardée dans un nouveau modèle d'instance.

Afin d'illustrer tout cela, nous allons maintenant présenter un exemple d'ordonnancement des différentes étapes d'un module d'enseignement. Ce module étant effectué par un groupe d'étudiants (de 1 à n).

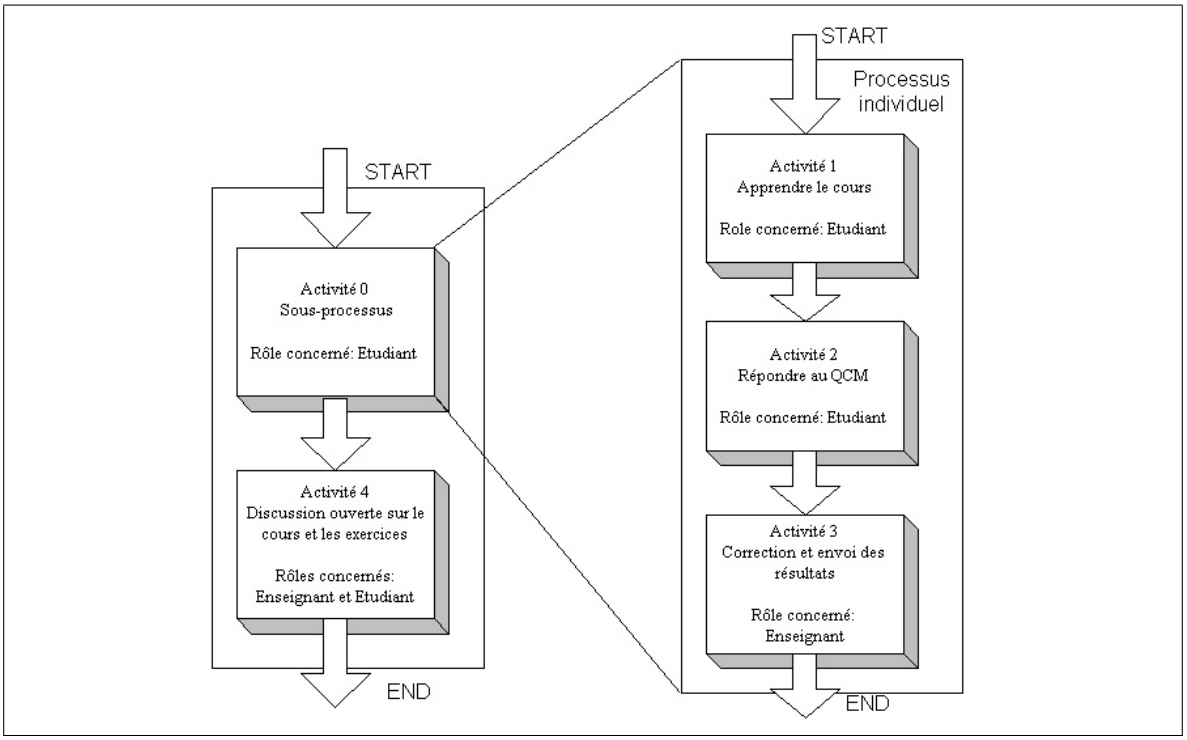


FIG. 4.2 – Exemple de modélisation

4.2.4.2 Description de l'exemple

Pour illustrer les principes de modélisation, nous allons prendre l'exemple d'un cours de physique. Ce module d'enseignement est décomposé en quatre étapes successives qui sont :

- l'activité *apprentissage du cours* associée au rôle *Étudiant* ;
- l'activité *réalisation d'exercices* associée au rôle *Étudiant* ;
- l'activité *correction* associée au rôle *Enseignant* ;
- l'activité *discussion sur le module* associée aux rôles *Étudiant* et *Enseignant*.

4.2.4.3 Modélisation

Un des paramètres à prendre en compte lors de la réalisation du modèle est l'individualisation du processus global d'apprentissage, i.e. la possibilité pour chaque étudiant d'avancer à son rythme dans certaines parties du processus.

L'enseignant doit décider de la manière dont les étudiants du groupe vont réaliser ces différentes activités. Nous pouvons distinguer deux grands modes de fonctionnement :

- Le premier mode consiste à dire qu'une activité est terminée uniquement lorsque tous les étudiants ont terminé. Cela signifie qu'un étudiant ne peut commencer l'activité *réalisation d'exercice* que lorsque tous les autres étudiants ont terminé l'apprentissage du cours. L'avancement est réalisé de manière synchrone pour tous les étudiants. Cette politique est très rigide. Elle est une copie presque parfaite des cours en présentiel.
- Le deuxième mode de fonctionnement consiste à déterminer des enchaînements d'activités qui peuvent être réalisés de manière autonome par chaque étudiant. Dans notre exemple, l'enseignant peut souhaiter que les étudiants apprennent le cours puis réalisent les exercices à leur rythme (et dans le respect des contraintes temporelles). De plus, il souhaite corriger les exercices au fur et à mesure que les étudiants envoient leurs résultats. Cela signifie qu'un étudiant qui a terminé l'apprentissage du cours peut réaliser les exercices sans attendre que les autres étudiants aient terminé la première activité. Cette politique correspond plus à la philosophie d'enseignement flexible associé à l'éducation à distance. L'étudiant évolue à son rythme à l'intérieur d'un groupe.

Pour supporter ces deux modes, nous utilisons des sous-processus. Dans notre exemple, l'enseignant décide que les trois premières étapes peuvent être réalisées de manière indépendante par chaque étudiant. L'enchaînement de ces activités réalisées de manière individuelle (cours, exercice, correction) est donc modélisé à l'intérieur d'un processus. Le processus global du module d'enseignement est alors composé de deux activités séquentielles, la première faisant référence au processus individuel, la deuxième étant l'activité de discussion qui est réalisée de manière synchrone entre tous les participants.

4.2.4.4 Exécution

Lors de la création du processus, nous passons en paramètre un modèle d'instance représentant les liens entre le modèle et les ressources (participants et documents). Par exemple, dans le modèle nous avons défini 2 rôles (enseignant et étudiant). Dans le modèle d'instance nous indiquons les personnes ayant le rôle enseignant et celles ayant le rôle étudiant. L'assignation peut être globale pour le modèle, ou n'être valable que pour une activité particulière. L'utilisation de modèles d'instances, nous permet une réutilisation des différents modèles existants, aussi bien pour les processus que pour les activités.

Le fonctionnement global est illustré figure 4.3. À partir des modèles de processus et d'activités ainsi que du modèle d'instance, le moteur de workflows va créer un sous-processus pour chaque étudiant. Ce sous-processus comporte les trois étapes Cours-Exercices-Correction. Chacune de ces activités ne contient qu'un seul workitem. Lorsque tous les sous-processus sont terminés, le moteur crée l'activité de discussion. Cette activité est composée de n workitems, un pour chaque étudiant et un pour l'enseignant.

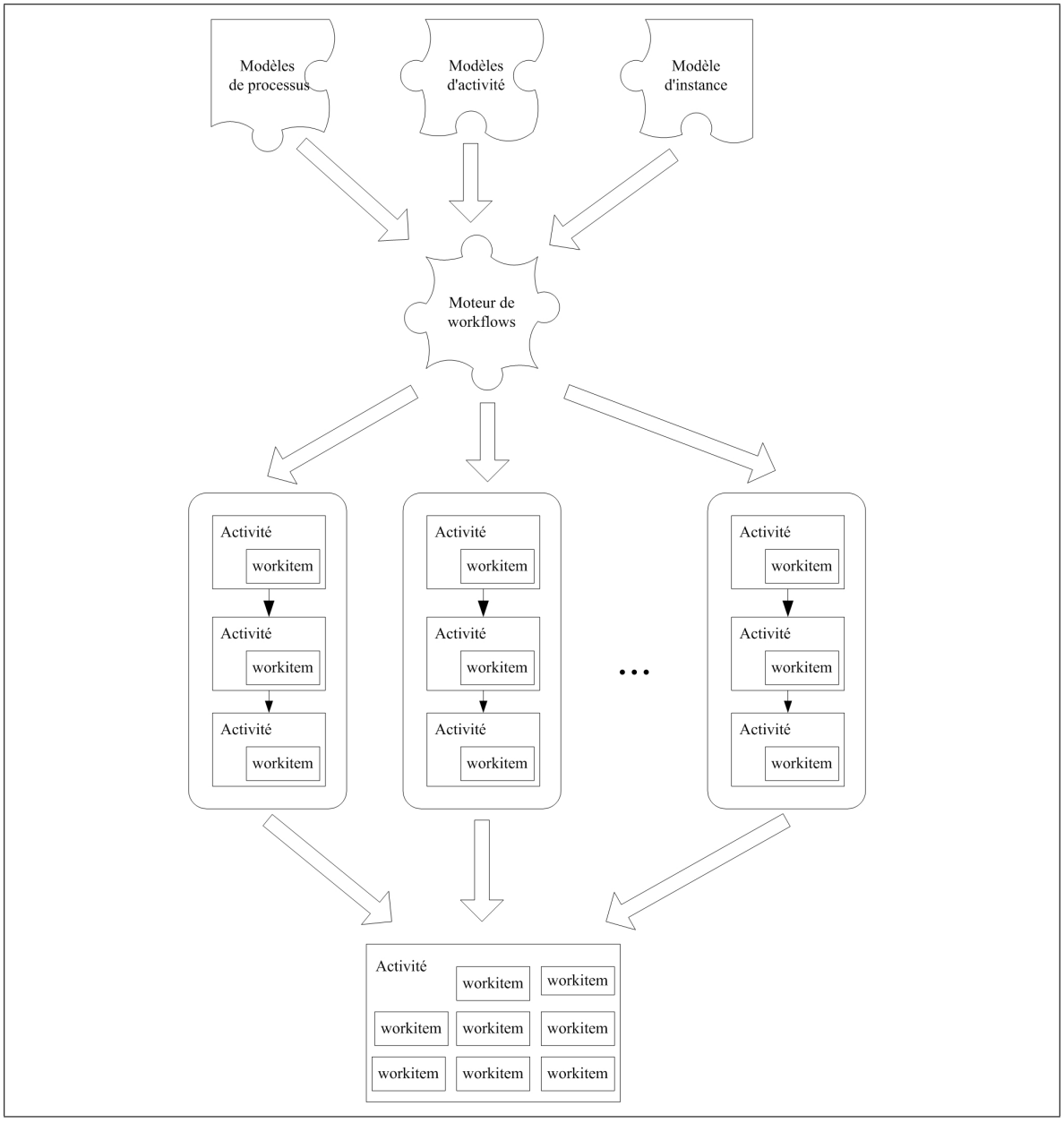


FIG. 4.3 – Instanciation des modèles

4.3 Technologies et concepts logiciels

Dans cette section, nous allons présenter les différentes technologies et les modèles de conception qui vont nous permettre de réaliser l'implémentation de notre système en gardant à l'esprit nos objectifs de flexibilité, de réutilisabilité et d'interopérabilité.

4.3.1 Les technologies à composants logiciels

Les composants logiciels sont une forme de programmation s'inspirant du succès des autres domaines industriels comme l'électronique et la mécanique. L'un des objectifs est la création d'une industrie du composant logiciel au même titre que les composants électroniques. L'utilisation de composants a pour objectif la création et la maintenance d'applications à partir de briques de bases interchangeables (les composants) que l'on pourrait acheter chez n'importe quel vendeur. Il ne resterait plus alors qu'à les assembler et les configurer. Cette approche présente donc l'immense avantage de la réutilisation de code déjà existant.

Pour des soucis d'intégration et plus particulièrement d'exécution de notre plate-forme sur des systèmes d'exploitations différents, notre choix s'est naturellement porté vers les modèles de composants proposés pour le langage Java. Nous allons maintenant les présenter plus en détail.

4.3.1.1 JavaBeans

Les JavaBeans, introduit par Sun en 1996, sont la première démarche composant liée au langage de programmation Java[8]. Les Enterprise JavaBeans (EJB) [DmYK01] sont la deuxième démarche composant autour de Java. Ces deux approches sont complémentaires.

Les composants JavaBeans supportent les modèles d'interaction événementiel et par appel de méthode. Toutefois si l'on peut envisager une communication à distance par appels de méthodes grâce à RMI (*Remote Method Invocation*), la communication événementielle reste strictement locale.

La norme JavaBeans propose des conventions de nommage concernant notamment l'accès aux attributs d'un composant. Ces conventions facilitent la vie du développeur. Par l'intermédiaire d'outils appropriés, on peut connaître sans difficulté les événements, les méthodes et les attributs gérés par un JavaBean.

4.3.1.2 Enterprise JavaBeans

La spécification de ces composants sépare la logique métier (ce que doit faire le composant) des propriétés non fonctionnelles (gestion de la sécurité, des transactions, *etc*). Une représentation schématique d'un serveur EJB est visible à la figure 4.4.

Un composant EJB est constitué de deux interfaces et d'une implémentation. L'interface *Home* est utilisée par les clients pour gérer le cycle de vie du composant (e.g. création, destruction, ...). L'interface *Remote* représente l'interface métier du composant. Ces deux interfaces sont implémentées par le *Bean*.

[8] http ://java.sun.com

Les EJB sont déployés dans un conteneur qui offre aux composants un ensemble de fonctions systèmes (transactions, persistance, support de la charge, ...) libérant ainsi le programmeur qui peut se concentrer sur le code métier. Le conteneur est un espace d'exécution au sein d'un serveur EJB.

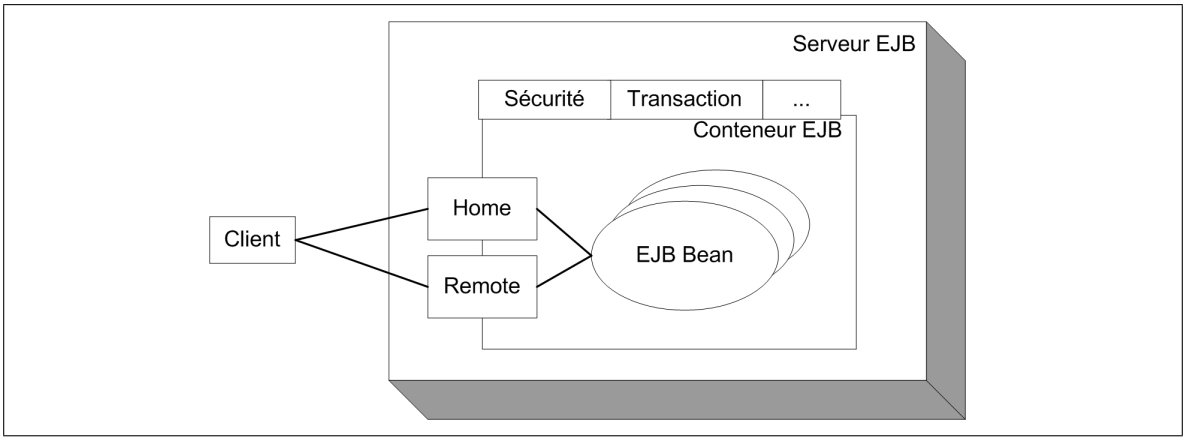


FIG. 4.4 – Représentation schématique d'un EJB

La dernière spécification [DmYK01] défini trois types d'EJBs que nous allons maintenant voir plus en détail.

Entity Bean. Les beans entités ont une durée de vie longue. Ils peuvent être partagés par de multiples clients et restent utilisables après un redémarrage du serveur. Ils représentent un ensemble de données persistantes du système d'informations, généralement sauvegardé dans un SGBD. La persistance peut être gérée soit manuellement soit automatiquement. Dans le cas des *Bean-Managed Persistence* (BMP), le développeur écrit lui-même les différentes requêtes SQL de sauvegarde et de récupération des données. Cette catégorie est principalement utilisée pour réutiliser des données existantes. Dans le cas des *Container-Managed Persistence* (CMP), le conteneur gère la persistance en utilisant le fichier de déploiement qui contient notamment les différents champs à sauvegarder.

Session Bean. Comme leurs nom l'indique, les beans de session ont une durée de vie contrainte à la durée de la session d'un client. Ils sont créés en réponse à une requête d'un seul client, communiquent exclusivement avec un seul client et disparaissent lorsque le client n'a plus besoin d'eux. Ainsi une bean de session est généralement utilisé pour fournir des services à un client particulier. Il existe deux types de bean de session, les *stateful beans* et les *stateless beans*. Un bean stateless ne garde aucune information d'un appel de méthode à l'autre, il n'utilise pas de variable d'instance globale. Ainsi deux instances d'un bean stateless sont rigoureusement identiques. Au contraire, un bean stateful possède des variables d'instances. L'exemple le plus classique est celui d'un panier d'achats d'un site de e-commerce. Le bean ne dure que le temps des courses virtuelles

d'un client et contient tous les éléments qu'il a choisi. Il peut en ajouter de nouveaux ou en retirer. Le serveur peut assurer une certaine persistance temporaire des informations.

Message-Driven Bean. Cette catégorie de composants est apparue dans la dernière spécification. Elle permet aux applications de traiter de manière asynchrone et anonyme les messages produits par l'API JMS (*Java Message Service*). Le MDB est un listener qui consomme les différents messages. Contrairement aux entités ou aux sessions il ne dispose pas d'interface home et remote. Il est dans un certain sens similaire à une session sans état. Ces instances ont une existance relativement courte et ne contiennent pas d'état d'un client spécifique. Il peut y avoir de nombreuses instances identiques fonctionnant au même moment.

Ce choix des EJB nous libère de la gestion des éléments non-fonctionnels et nous permet de répondre en partie à notre problème de support à la charge. En effet de nombreux serveurs d'EJB supportent le clustering afin de répartir la charge sur différentes machines de manière transparente pour le développeur et pour l'utilisateur. Nous pouvons ainsi ajouter des serveurs pour faire face à l'affluence de nouveaux apprenants ou en retirer lors d'une baisse d'activité.

4.3.2 Les technologies d'interopérabilité

Comme nous l'avons indiqué dans le premier chapitre, nous ne souhaitons pas dans le cadre des travaux sur COW recréer complétement une plate-forme d'enseignement à distance, mais uniquement offrir COW comme un des composants de ces plates-formes. Pour assurer le fonctionnement de notre système avec un maximum de plates-formes existantes, nous souhaitons limiter les contraintes technologiques en favorisant les protocoles standardisés et reconnus. Pour réaliser l'interopérabilité entre systèmes distribués, nous disposions de deux solutions majeures, le bus objet CORBA et les services web avec le protocole SOAP. La première solution pourrait sembler la plus naturelle car nous avons choisi d'implémenter une spécification de l'OMG reposant naturellement et nativement sur CORBA. Cependant ce bus logiciel pose quelques problèmes dans le cas de communication inter-réseaux (inter-entreprises), car la plupart des routeurs entre les deux points/systèmes bloquent, pour des raisons de sécurité, les ports et/ou le protocole IIOP utilisé pour le transport des informations. La deuxième solution, plus récente présente l'avantage d'être basée sur les protocoles de communication standards de l'internet (HTTP et/ou SMTP) et sur des langages textuels basés sur XML plus facilement manipulable. De plus, le nombre de langage supportant le protocole SOAP est beaucoup plus important.

Nous allons maintenant détailler un peu plus le mode de fonctionnement des services web. Le lecteur intéressé trouvera une description complète et détaillée dans l'article [Riv02].

4.3.2.1 SOAP

Simple Object Access Protocol (SOAP) [W3C00, Riv02] est un protocole basé sur l'envoi et la réception de messages. Il est basé sur XML.

L'envoi et la réception de messages sont assurés par des protocoles de niveau inférieur. Les deux plus utilisés sont HTTP et SMTP.

4.3.2.2 WSDL

Web Service Description Language (WSDL) [W3C01] permet de spécifier les interfaces d'un service web, i.e., les différentes opérations réalisables et leurs localisations. Il joue un rôle similaire au langage IDL de CORBA. Ce langage est complètement indépendant des différents langages de programmation, et dans une certaine mesure, il est même indépendant du protocole SOAP.

4.3.2.3 UDDI

Le service *Universal Description, Discovery and Integration* (UDDI) [UDD] est utilisé pour la publication des interfaces exprimées en WSDL. Il sert à la fois de service de nommage et de service de courtage.

4.3.2.4 Conclusion

SOAP ainsi que les autres protocoles et langages de services associés présentent un double intérêt pour notre plate-forme. Ils vont nous permettre d'utiliser facilement et d'intégrer des outils externes avec notre moteur, mais ils vont surtout permettre aux développeurs souhaitant utiliser notre système de workflows de l'intégrer facilement dans leurs futurs applications. Ils ne devront plus se poser la question du langage à utiliser et de l'OS. Nous ne leurs imposons pas ainsi l'utilisation du langage et de la plate-forme que nous avons choisie. Nous verrons par la suite des exemples de réalisations utilisant SOAP pour s'interfacer avec COW.

4.3.3 Patrons de conception

Les patrons de conception (*design patterns*) décrivent des problèmes qui arrivent fréquemment et nomment, extraient et identifient les aspects clés d'un structure de conception [GHJV95] [AIS+77]. Ils permettent aux développeurs de trouver facilement et rapidement des solutions de conception éprouvées. Cela augmente la facilité de maintenance en fournissant une vision claire des choix de conception et une meilleure identification des fonctionnalités des différents composants.

Nous allons maintenant présenter brièvement les principaux patrons qui offrent des solutions à nos problèmes. Une description détaillée se trouve dans [GHJV95].

4.3.3.1 Patron Strategie

Ce patron définit une famille d'algorithmes, encapsule chacun d'eux et les rend interchangeables. Cela permet de sélectionner l'algorithme qui répond le mieux aux besoins de chaque instance. L'intérêt majeur est d'offrir une grande souplesse du code. Ce patron évite d'avoir dans le code la formulation d'un choix conditionnel monolitique. Dans notre cas cela nous permettra d'offrir à l'utilisateur la possbilité de modifier le comportement du moteur. De plus cette approche nous permet d'ajouter facilement d'autres comportements afin de répondre aux nouveaux besoins des utilisateurs et ainsi faire évoluer de manière continue notre moteur.

4.3.3.2 Patron Façade

Ce patron fournit une interface unifiée à un ensemble d'interfaces du sous-système. Une façade définit une interface de haut niveau qui rend le système sous-jacent plus facile à utiliser. Grâce à ce patron nous pouvons ajouter des fonctionnalités de plus haut niveau sans toucher au cœur de notre système.

4.3.3.3 Patron Adaptateur

Ce patron convertit la classe d'une interface en une autre interface attendue par le client. Les adapteurs permettent à des classes de travailler ensemble. Ce patron nous intéresse plus particulièrement pour l'intégration et l'interopérabilité de notre système dans des plates-formes existantes.

4.3.3.4 Patron Type Object (*Type Object*)

Ce patron utilise deux classes séparées pour les instances et pour leurs types [JW97]. Dans le contexte d'un système de workflows, ce patron représente le noyau de la mise en œuvre du modèle. Les règles métiers sont localisées du côté du type, tandis que les informations d'exécution sont situées du côté de l'instance. Ce patron résout certains problèmes liés à l'implantation d'un système de workflows [MJ98] :

- Il permet de créer de nouveaux objets types à l'exécution. Elles sont des instances de la classe Type. Comme dans de nombreux frameworks orientés objet, de grandes possibilités de personnalisation sont possibles lors de la création des nouvelles instances.
- Les objets représentant le Type emploient le patron fabrique [GHJV95] pour créer des instances. La fabrique peut personnaliser chaque instance créée. Il est ainsi possible d'obtenir une large variété d'instances pour un même type.
- Ce découpage est invisible pour les humains et les applications interagissants avec le système de workflows. La partie Type implante des fonctions communes alors que la partie instance implante des fonctions spécifiques à l'exécution.
- Le lien entre instance et type est sous contrôle. Il est ainsi possible de changer dynamiquement le type d'une instance lors de l'exécution. Cela permet aux utilisateurs de modifier le comportement des instances.

4.3.4 Réflexivité et implémentation ouverte

La réflexivité est définie en informatique comme la capacité d'un programme de manipuler comme des données une représentation de son propre état durant son exécution et de le modifier [BGW93] [Mae87]. Ces manipulations concernent deux aspects : l'introspection et l'intercession. L'introspection est la capacité d'un programme d'observer et de raisonner sur son propre état. L'intercession est la capacité pour un programme de modifier son propre état d'exécution ou de transformer sa propre interprétation ou signification. Ces deux aspects nécessitent des mécanismes pour encoder les états d'exécution comme des données ; la fourniture d'un tel encodage est appelé réification.

Dans le cadre de nos travaux, le concept de réflexivité se retrouve à deux niveaux d'abstraction différents. Le premier concerne les modèles de processus et d'activité que nous allons exécuter. Ces modèles sont exprimés dans un langage que nous pouvons rapprocher d'un langage de programmation simpliste. Offrir de l'introspection pour ce langage cela consiste à permettre la visualisation des différents consistuants et de leurs attributs et ceci même s'ils ne sont pas encore instantiés (Nous utilisons en effet un mécanisme d'instantiation tardif, ou plus précisément à la demande). Le modèle n'est pas complètement instantié lors du lancement d'un processus. Cela nous permet d'éviter de créer des instances d'activités qui pourront n'être jamais utilisées par la suite car elles se trouveront sur un chemin d'exécution qui ne sera pas emprunté. Offrir de l'intercession pour ce langage cela consiste à permettre une modification dynamique, afin par exemple d'ajouter une nouvelle activité ou de retirer un chemin d'exécution devenu obsolète. L'introspection est de fait un élément fondamental pour la flexibilité des moteurs de workflows. La réflexivité est également un moyen d'obtenir l'adaptabilité des systèmes de worklows [EtH98].

Le deuxième niveau qui nous intéresse est la plate-forme d'exécution elle-même. L'introspection permet de connaître son comportement et l'intercession nous permet de le modifier. La réflexivité à ce niveau dépend du langage de programmation utilisé pour l'implémentation. Dans notre cas, comme nous utilisons le langage java, nous devons « recréer » l'intercession car il n'est pas possible de dynamiquement modifier l'implémentation d'une classe. Pour réaliser cela, nous nous sommes basés sur le patron stratégie en identifiant les points de modification dans notre code.

Cette démarche d'ouverture des composants de notre système de telle manière que l'utilisateur puisse participer à la sélection de la stratégie est appelée implantation ouverte (*open implementation*) [Kic96] [KLL+97] [MLMK97]. Cette approche définit quatre styles d'interfaces :

Style A. Il n'existe pas d'interface de contrôle de la stratégie d'implantation. Cela correspond à l'abstraction boîte noire classique.

Style B. Le client fournit une description de son usage du comportement. Cela correspond à paramétrer le fonctionnement.

Style C. Le client choisi la stratégie d'implantation parmi une liste des choix disponibles.

Style D. Le client fournit lui-même la stratégie à utiliser.

Le style D est pour nous le plus intéressant car il offre l'avantage d'externaliser le comportement de notre moteur, nous permettant d'adapter notre moteur aux différents souhaits des utilisateurs.

Après avoir présenté les différents choix de modélisation et de technologies que nous avons réalisés, nous allons maintenant passer à leur mise en œuvre pour la construction de COW, notre moteur de workflows flexible.

4.4 Mise en œuvre

Dans cette section nous allons présenter l'implantation de notre moteur de workflow.

4.4.1 Architecture

Afin d'atteindre nos objectifs d'extensibilité et de réutilisabilité, nous avons choisi de modulariser au maximum la réalisation de COW. Cette conception nous offre de plus une plus grande facilité de maintenance et d'évolution des fonctionnalités.

La figure 4.5 représente l'architecture actuelle de notre moteur. Celle-ci est décomposée en quatres domaines distincts :

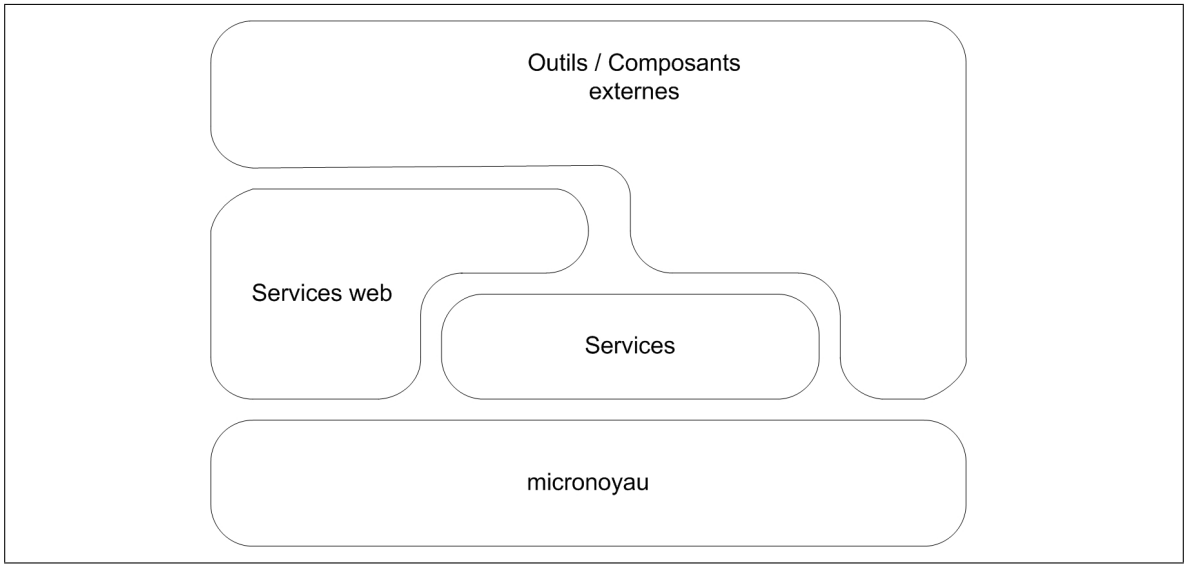


FIG. 4.5 – Architecture globale de COW

- **Le micronoyau** représente le cœur de notre système. Nous avons suivi une démarche proche de celle réalisée dans le domaine des systèmes d'exploitation à micronoyau [Tan94]. La philosophie de ce type d'architecture est d'offrir un ensemble minimale de fonctions spécifiques. Cette minimalité rend cette architecture très souple car implique peu de contraintes pour ajouter des fonctionnalités dédiées de plus haut niveau. Dans le cadre d'un système de workflows les fonctionnalités de bases concernent la gestion rudimentaire des processus et des activités, la circulation des documents et l'envoi/réception d'événements permettant au mode extérieur de connaître les opérations en cours du système. Cette approche minimaliste a également été employée pour la conception de micro-workflow [Man00].
- **Les services** offrent un niveau d'abstraction supplémentaire par rapport au noyau en offrant des services de plus haut niveau avec une valeur ajoutée pouvant varier. Certains services n'offrent qu'une vue unifiée et simplifiée du micronoyau. Ils sont une mise en œuvre du patron `façade` [GHJV95]. D'autres plus complets permettent par exemple de gérer la liste des travaux d'un utilisateur.
- **Les services webs** offrent une couche d'abstraction et de transformation permettant l'intégration du micronoyau et des services dans un environnement hétérogène. Ils réalisent notamment des conversions de types comme par exemple la transformation d'ob-

jets Java en une représentation XML. Cela nous permet de n'utiliser que des types de données simples pour la communication SOAP. Les types de données complexes sont beaucoup plus difficiles à faire circuler car ils nécessitent souvent une sérialisation puis une désérialisation qui ne sont pas toujours réalisables de manière simple entre deux mondes technologiques différents comme par exemple pour passer du monde Java au monde Visual Basic. Une partie de ces services est basée sur le patron adaptateur.

– **Les outils et composants externes** accèdent et manipulent les différents services offerts par la plate-forme. Dans ce niveau nous allons retrouver les outils de surveillance, de manipulation des modèles et de leurs instances, ainsi que toute la gestion des interactions des utilisateurs avec notre système.

Nous allons maintenant voir plus en détail l'implémentation du micronoyau.

4.4.2 Le micronoyau

Comme nous l'avons vu dans la section précédente, le micronoyau est le cœur de notre système. Nous avons choisi comme point de départ pour son implantation, l'utilisation de la spécification WMF de l'OMG présentée dans le chapitre 3. Comme nous l'avons expliqué, notre architecture repose sur l'utilisation des composants EJB. Notre premier travail fut la transcription dans cette technologie du WMF.

4.4.2.1 Passage des objets CORBA aux composants EJB

Afin de mieux comprendre le fonctionnement de la spécification, nous avions réalisé au début de nos travaux une première implantation en utilisant des objets CORBA. Pour faire face à certains problèmes, notamment la persistance, nous avons décidé de migrer vers les EJB. Les deux réalisations étant en Java, nous souhaitions réutiliser un maximum de code existant. Cela nous a amené à chercher les liens possibles entre les deux technologies (tableau 4.1).

CORBA	EJB
Interface IDL	Interface Remote
Implémentation	Classe Bean
Fabrique supplémentaire	Interface Home

Tab. 4.1 – Correspondance entre technologie CORBA et EJB

Les interfaces IDL CORBA qui décrivent les services fournis par un objet correspondent aux interfaces Remote des EJB. Afin de gérer le cycle de vie des objets CORBA, nous avions ajouté un ensemble de fabriques [VP01] Nous les avons fait disparaitre, le conteneur EJB se chargeant dorénavant de ce travail. Les interfaces des fabriques ont été reprises et transformées pour respecter les standards de nommage des interfaces Home des EJB. Une partie de l'implémentation des fabriques a pu être réutilisée dans les méthodes EJBCreate des Beans.

Enfin, comme nous utilisions précédemment le langage Java, la majorité de l'implémentation des méthodes existantes a été réintégrée avec peu de modifications.

Cette migration s'est réalisée relativement vite (environ un mois) bien que nous n'ayons jamais utilisé les composants EJB précédemment. Ce changement de technologie nous apporte de plus une simplification de nos développements en nous libérant de la gestion du cycle de vie et de la persistance.

4.4.2.2 Expression des modèles : passage de XML aux objets

Nous avons vu au début de ce chapitre le méta-modèle que nous utilisons ainsi que l'expression des modèles en XML. Nous devons, dans notre moteur, gérer les modèles et assurer l'instantiation correcte des différents composants correspondants. La spécification WMF nous impose peu de contraintes et n'oblige pas l'utilisation d'un langage particulier. Un point intéressant du WMF est l'utilisation d'un objet `WfProcessMgr`. Chaque instance de cet objet est dédiée à la gestion d'un modèle particulier de processus. Cette décomposition permet de considérer ce composant comme le type d'un objet et le `WfProcess` comme une instance de ce type. Cela correspond au patron de conception *TypeObject* [JW97].

Concernant les activités, le WMF laisse le champ libre au développeur. Comme nous l'avons expliqué au début du chapitre, chaque activité dispose de son propre modèle de la même manière qu'un processus. Nous avons donc ajouté un composant `WfActivityMgr`, basé sur le `WfProcessMgr` et réalisant le même type de fonction pour gérer les activités. La conception et le comportement du moteur sont ainsi rendus plus cohérents, les activités et les processus sont traités de la même manière.

Dans notre modélisation, nous avons ajouté en plus du processus et de l'activité un troisième niveau représenté par le concept de tâche. Ce dernier n'existe pas dans le WMF. Nous l'avons ajouté par l'intermédiaire du composant `WfTask` qui hérite de `WfExecutionObject`. Cet ajout nous à obligé à redéfinir les liens avec le composant `WfAssignment`, qui est maintenant lié aux `WfTask`. Tous ces ajouts et modifications sont représentés dans le diagramme UML de la figure 4.6.

Nous allons maintenant voir plus en détail les différentes phases de création et d'utilisation du moteur de workflows.

4.4.2.3 Gestion du flux de contrôle

La structuration des activités et leurs ordonnancement sont la caractéristique essentielle d'un système de workflows. Le WMF ne s'intéresse pas du tout à cette partie en laissant le développeur libre de faire ses propres choix. Nous avons imposé comme contrainte qu'un modèle de processus comporte au minimum une activité dont l'identifiant sera obligatoirement START. De même, la dernière activité d'un processus aura l'identifiant END. Cependant la présence de cette dernière activité n'est pas obligatoire. Cela permet d'ajouter continuellement de nouvelles activités à un processus. Cette approche est particulièrement intéressante dans le cas de la constitution dynamique d'un étudiant où l'on connaît toujours son point de départ mais pas son point d'arrivée. Les autres activités peuvent avoir n'importe quel identifiant.

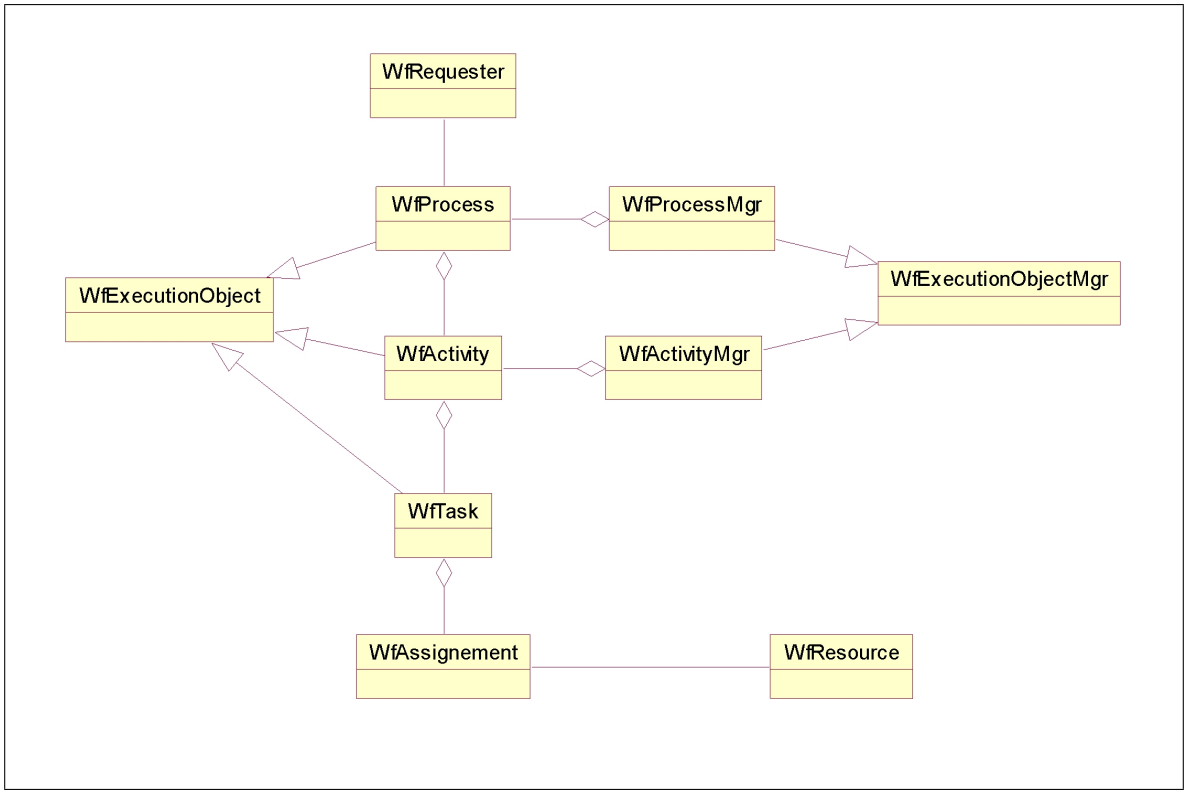


FIG. 4.6 – Diagramme UML des changements du WMF

La gestion du flux de contrôle est réalisée par un objet de type `ModelManager`. Cet objet manipule un ensemble d'objets représentants les différentes entités liées au modèle de processus (`ActivityLink`, `Control Flow`, et les conditions associées). A chaque changement d'état d'une activité, il fournit au `WfProcess` la liste des `WfActivity` à créer. L'algorithme de choix est le suivant :

NEXT_ACTIVITIES()

```
1  if PostCondition == Split ET
2    then for chaque transition_sortante == true
3              do
4                 Vérification des préconditions de l'activité d'arrivée
5                 if prcondition == true
6                   then ajouter Id de l'activité dans la liste des résultats
7    else  if PostCondition == SplitXOR
8            then j = 0; i = false
9                 while i ≠ true & j < nb transition sortantes
10                   do if transition_i
11                         then Vérification des préconditions de l'activité d'arrivée
12                              if précondition == true
```

```
13                                  then ajouter Id de l'activité dans la liste des résultats
14                                       i = true
15                                  else j++
16      else error "Type inconu"
17           return Tableau des activités
```

Chaque transition dispose d'un état qui peut être `true` (la transition est valide) ou `false` (la transition est invalide). L'état d'une transition est déterminée par les conditions qui y sont attachées. S'il n y a pas de condition, la transition est supposée valide. Nous n'avons pas imposer de langage particulier pour l'expression des conditions. Chaque élément représentant une condition contient un champ déterminant la stratégie à appliquer. Il est ainsi possible d'utiliser différents langages à l'intérieur d'un même processus. Le même traitement est appliqué au pré et post -conditions des `Activity Link`. Avec la liste des activités qu'il reçoit, le Wj Process va chercher chaque `WfActivityMgr`, lui demander la création d'une nouvelle instance d'activité. Puis il va initialiser l'activité (cf section suivante sur le flux de données) et la démarrer.

Lorsqu'une activité est créée, elle réagit en fonction de son type. Si elle est de type activité de routage, elle va automatiquement se terminer et envoyer un événement afin de prévenir le processus. Si l'activité est de type sous processus, elle va rechercher le `WfProcess` correspondant et demander la création d'une nouvelle instance de processus, et la démarrer. Lorsque ce processus se termine, il prévient l'activité qui a demandé sa création. Cette activité peut alors se terminer.

Si l'activité est de type implémentation, elle va créer les différentes `WfTask`. Plusieurs cas peuvent se produire :

- Le cas le plus simple est la création d'une tâche pour chaque personne détenant le rôle spécifié.
- La tâche est assignée à plusieurs personnes

La politique par défaut de ce type d'activité est d'attendre la fin de toutes les tâches pour s'arrêter. Cette stratégie est modifiable (pattern stratégie).

4.4.2.4 Gestion du flux de données

L'autre aspect important d'un système de workflows est la circulation des différentes données nécessaires au bon fonctionnement des différents constituants (processus, activité, tâche et conditions). Dans le méta-modèle, cette circulation est représentée par l'entité `Data flow`. La gestion des données est laissée principalement au choix du développeur par le WMF. La spécification définit des méthodes pour le WfProcessMgr. La première, `get_context_signature` permet de récupérer un tableau d'objet de type `NameValueInfo`. Ce type comprend deux variables, une déterminant le nom de la donnée et une décrivant le type de la donnée. Il est ainsi possible de connaître les différents paramètres attendus en entrée d'un processus. L'autre méthode, `get_result_signature`, renvoie également à un tableau de `NameValue` contenant les descriptions des sorties du processus. Ces deux méthodes sont présentes dans le WfActi-

vityMgr que nous avons ajouté. Elles nous permettent ainsi de réaliser un contrôle des types de données lors de la circulation des informations.

L'objet WfExecutionObject contient deux méthodes (`process_context` et `set_process_context`) qui permettent de récupérer le contexte d'exécution (les données en entrée) ou de l'initialiser. Chaque objet WfProcess et WfActivity et son extension WfTask possèdent deux méthodes (`result` et `set_résult`) qui représentent le résultat produit. Les quatre précédentes méthodes utilisent comme valeur d'entrée ou de sortie des tableaux de `NameValue`. Ces objets sont composés de deux variables, une représentant le nom et une représentant la valeur de la donnée.

Maintenant que nous savons où trouver les informations relatives aux données du workflow, il ne nous reste plus qu'à les faire circuler. Pour cela, et sur le même principe que pour la gestion du flux de contrôle, nous avons développé un objet `DataFlowManager` qui gère un ensemble d'objets de type `DataLink` représentant la circulation d'une donnée. Après la création d'une activité, le processus interroge le `DataFlowManager` afin de connaître les différents liens de données entrant dans cette activité. Pour chacun de ces liens, le processus récupére les données des activités d'origines (méthode `result`), il extrait la donnée qui l'intéresse pour le transfert et l'ajoute dans le tableau de données entrantes. Une fois toutes ces opérations effectuées, le tableau permet d'initialiser l'activité (méthode `set_context`) et elle peut ainsi démarrer. Dans la majorité des cas, concernant les modèles qui nous intéressent, les données sont contenues dans un tableau. Par exemple une activité d'évaluation va créer une tâche pour chaque étudiant, le résultat global sera donc un tableau contenant les réponses de chaque étudiant.

4.4.2.5 Gestion de l'historique

Le WMF présente un certain nombre d'événements pour surveiller et contrôler le fonctionnement du moteur. Nous avons ajouté quelques événements spécifiques à notre implantation comme par exemple `WfCreateActivityEventAudit` pour avertir de la création d'une nouvelle activité. Lors de sa création, un événement workflow produit un message JMS afin d'alerter les différentes applications connectées à notre système. Cela permet d'avoir des clients en fonctionnement PUSH. Chaque événement est persistant et n'est pas détruit avec la fin et la destruction du processus qui les a générés.

4.5 Conclusion du chapitre

Dans ce chapitre, nous avons rappelé les principaux objectifs de notre plate-forme qui sont une plate-forme centrée sur l'utilisateur où les modèles sont au cœur du système et où des mécanismes de flexibilité permettent de modifier dynamiquement le modèle d'exécution ainsi que le comportement du moteur. Nous avons ensuite fixé notre cadre de travail (intégration, respect des standards, réutilisation et généralisation, extensibilité, fonctionnement synchrone et asynchrone et le support de la charge). Afin de répondre aux objectifs dans le cadre fixé, nous avons suivi une approche par méta-modèle et utilisé une technologie à composants logiciels. Nous avons présenté notre architecture en couche, composé d'un micronoyau qui regroupe

les fonctionnalités de base de notre moteur, des façades qui offrent des services de plus haut niveau et des différentes applications externes qui utilisent la plate-forme.

L'implantation réalisée est constituée de 21 EJB entité, 12 EJB session et d'une quarantaine de classes, ce qui représente environ 42000 lignes de codes. Un prototype d'intégration avec le « Campus Virtuel » a également été réalisé par Archimed. Ces développements ont été placés en Open Source sous licence LGPL et hébérgés sur la forge du consortium ObjectWeb.

Nous avons également réalisé des adaptateurs (`ToolAgent`) afin de communiquer avec des outils externes. Nous avons réalisé un adaptateur permettant de lancer une application de QCM constituées de Servlet et d'EJB. Nous sommes capables de la lancer et de récupérer les résultats de l'évalution et de les utiliser notamment pour déterminer la suite du chemin de l'apprenant. Une autre réalisation est en cours afin de pouvoir communiquer avec SPIN, un CVE[9] développé par l'équipe GRAPHIX du Laboratoire d'Informatique Fondamentale de Lille (LIFL). La communication est dans ce cas là réalisé par l'envoi et la réception de messages SOAP.

Dans le prochain chapitre, nous allons nous focaliser sur les détails de notre moteur et plus particulièrement sur les aspects concernant la flexibilité et la gestion du temps.

[9] **C**ollaborative **V**irtual **E**nvironment

Chapitre 5

Détails d'architecture de COW

> The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!" (I found it) but "That's funny ..."
>
> **Isaac Asimov**.

Le chapitre précédent nous a permis de présenter l'architecture globale de notre système en insistant sur les aspects modélisation, technologies utilisées et leurs mise en œuvre dans le cadre de notre architecture à micronoyau. Dans ce chapitre, nous allons nous concentrer sur les détails de notre réalisation et plus particulièrement sur les aspects liés à la flexibilité et à la gestion du temps. Nous commencerons dans la section 5.1 par identifier les différents opérations de flexibilité d'un système de support à l'exécution de scénarios pédagogiques. Nous verrons ces opérations selon deux aspects différents, les aspects liés aux modèles et les aspects liés à la plate-forme d'exécution. Nous verrons en détail les solutions que nous apportons pour répondre à ces attentes dans la section 5.2. Comme annoncé précédemment, la gestion du temps est un élément incontournable. Nous verrons dans la section 5.3 les méthodes que nous proposons pour inclure le temps dans notre système de workflows. La section 5.4 présentera l'architecture que nous avons testée concernant les aspects utilisateurs et notamment un accès multi-canal à notre plate-forme. Enfin nous concluerons ce chapitre dans la section 5.5.

5.1 Support de la flexibilité

Comme nous l'avons vu dans le chapitre 3, la flexibilité des systèmes de workflows couvre deux grandes catégories, la gestion des exceptions et l'adaptativité des modèles. De nombreuses méthodes ont été développées afin de répondre aux besoins des différents domaines d'applications (financier, médical, ...). Avant de nous lancer dans la conception et le développement du support de la flexibilité dans notre moteur, nous allons commencer par déterminer quels sont les opérations dans le domaine de l'éducation à distance et plus particulièrement pour l'exécution de scénarios pédagogiques.

5.1.1 Identification des opérations sur les modèles

Si l'on se place dans le cas le plus courant, le modèle de processus est créé par un ingénieur pédagogique en lien avec un enseignant. Ce modèle est en théorie complètement défini. Lors de son exécution, nous avons identifié les scénarios suivants :

Raffinement de l'activité. Deux cas de figure peuvent se présenter. Dans le premier cas, le travail spécifié dans l'activité est trop important et il a besoin d'être redécoupé afin d'être mieux traité par les apprenants. Ce découpage peut se réaliser par ajout de tâches dans l'activité définissant chacune une partie des actions à réaliser. Dans ce cadre, le tuteur peut suggérer un ordre d'exécution mais les apprenants sont libres de le respecter ou non. Si l'ordonnancement est important, il est alors nécessaire de modifier le type de l'activité, qui va maintenant lancer un sous-processus correspondant à la nouvelle séquence d'opérations. Ce nouveau processus peut ne pas être complètement défini lors de sa création. Il sera à son tour raffiné. Cela nous amène à notre deuxième cas de figure qui est l'inverse du précédent. Des activités peuvent être sur-spécifiées. Le découpage est trop restrictif. Une tâche correspond à une simple opération sur un outil. Pour ajouter un peu plus de liberté, nous devons être en mesure de retirer des tâches, voir des activités au sein d'un processus.

Modification de la structure du processus. L'ajout et le retrait d'activités nécessitent une redéfinition de la structure du processus, aussi bien du flux de contrôle que du flux de données. Le système doit donc être capable d'ajouter des transitions, d'en enlever et d'en modifier. La modification peut concerner aussi bien la source de la transition que sa destination ou les différents éléments qui lui sont attachés (expression des conditions, ...).

Individualisation. Au cours d'une formation, il arrive que des apprenants s'écartent des autres membres du groupe, soit parce qu'ils éprouvent des difficultés, soit parce que leurs niveaux sont supérieurs. Dans ce cas, le tuteur doit pouvoir intervenir sur l'exéuction pour adapter une partie du processus global aux besoins spécifiques de ces apprenants, tout en les laissant dans leur groupe de formation.

Dans d'autres modules de formation où l'on souhaite que l'apprenant participe à la création de son cheminement, nous nous trouvons dans le cas suivant : la première activité consiste à définir la suite du processus. Une fois cette définition réalisée, ce processus est instantié.

Si nous dressons un bilan des modifications nécessaires pour adapter les modèles nous devons retenir les options suivantes :

- Possibilité d'ajouter, retirer et modifier des processus ;
- Possibilité d'ajouter, retirer et modifier des activités ;
- Possibilité d'ajouter, retirer et modifier des tâches ;
- Possibilité d'ajouter, retirer et modifier des transitions (flux de contrôle et flux de données) ;

Pour chacune des modifications que nous allons réaliser, nous devons prendre en compte la portée des changements, i.e., quelles sont les instances qui subiront les modifications. la portée peut être [GAC+97] : limitée à une instance (pour un étudiant particulier par exemple), limitée

à un groupe d'instances (un ensemble d'étudiants réalisant chacun un processus individuel) ou pour toutes les instances du modèle en cours d'exécution.

A cela s'ajoute la durée de la modification, affecte-t-elle uniquement une instance en cours ou également toutes les futures instances ? Dans notre cas, nous avons choisi d'offrir les deux possibilités. Chaque modification provoque la création d'un nouveau modèle qui pourra être de nouveau instantié dans le futur.

5.1.2 Identification des points ouverts de la plate-forme

Après avoir présenté les différents besoins de flexibilité liés au modèles et à leurs instances, nous allons maintenant voir les besoins de modification du comportement de la plate-forme elle-même. Comme nous l'avons indiqué dans le chapitre précédent, nous sommes dans l'incapacité de prévoir l'ensemble des comportements possibles que devrait avoir notre moteur afin de répondre à toutes les demandes présentes et futures. L'utilisation d'une approche implémentation ouverte basée sur le style D (cf section 4.3.4 ou [KLL+97]), nous semble la meilleure solution. Cependant, il est nécessaire de déterminer les points ouverts de notre architecture, i.e., les endroits où nous externaliserons le comportement.

La flexiblité du comportement de COW repose principalement sur l'utilisation du patron stratégie [GHJV95]. Dans la mesure où ne souhaitons pas reposer sur des fonctionnalités spécifiques d'un serveur d'application particulier, la limite de notre approche réside dans le cas où il n'est pas possible d'implanter une stratégie particulière, il est nécessaire d'effectuer des changements directement dans les composants du moteur. Cette catégorie de modification nécessite l'arrêt du serveur.

Afin de mettre en œuvre le patron stratégie, nous avons identifié les points ouverts permettant d'ajuster le comportement du micronoyau et nous les avons indiqués pour les composants majeurs :

Pour les processus : politique de gestion du temps, politique de détermination des activités suivantes, politique de création des activités.

Pour les activités : politique de gestion des tâches (validation de l'activité si toutes les tâches sont réalisées ou si un certain nombre de tâches sont terminées, ...), politique de gestion du temps.

Pour les tâches : politique de gestion du temps.

Pour les assignations : politique de découverte et d'affectation des outils ainsi que des personnes humaines.

Nous verrons plus en détail la gestion du temps dans la section 5.3.

Le dernier mécanime d'extension que nous proposons s'appuie sur le fonctionnement événementiel du moteur. Ce dernier produit des événements à chaque changement qui s'opére (création d'une activité, fin d'une tâche, modification d'un processus, ...). Les applications externes peuvent ainsi connaître les différentes opérations qui sont réalisées et réagir en conséquence. Ces applications peuvent être également une façon d'ajouter des fonctionnalités au moteur. D'un point de vue technologique, ces événements sont émis sous forme de message JMS.

5.2 Mise en œuvre de la modification des modèles

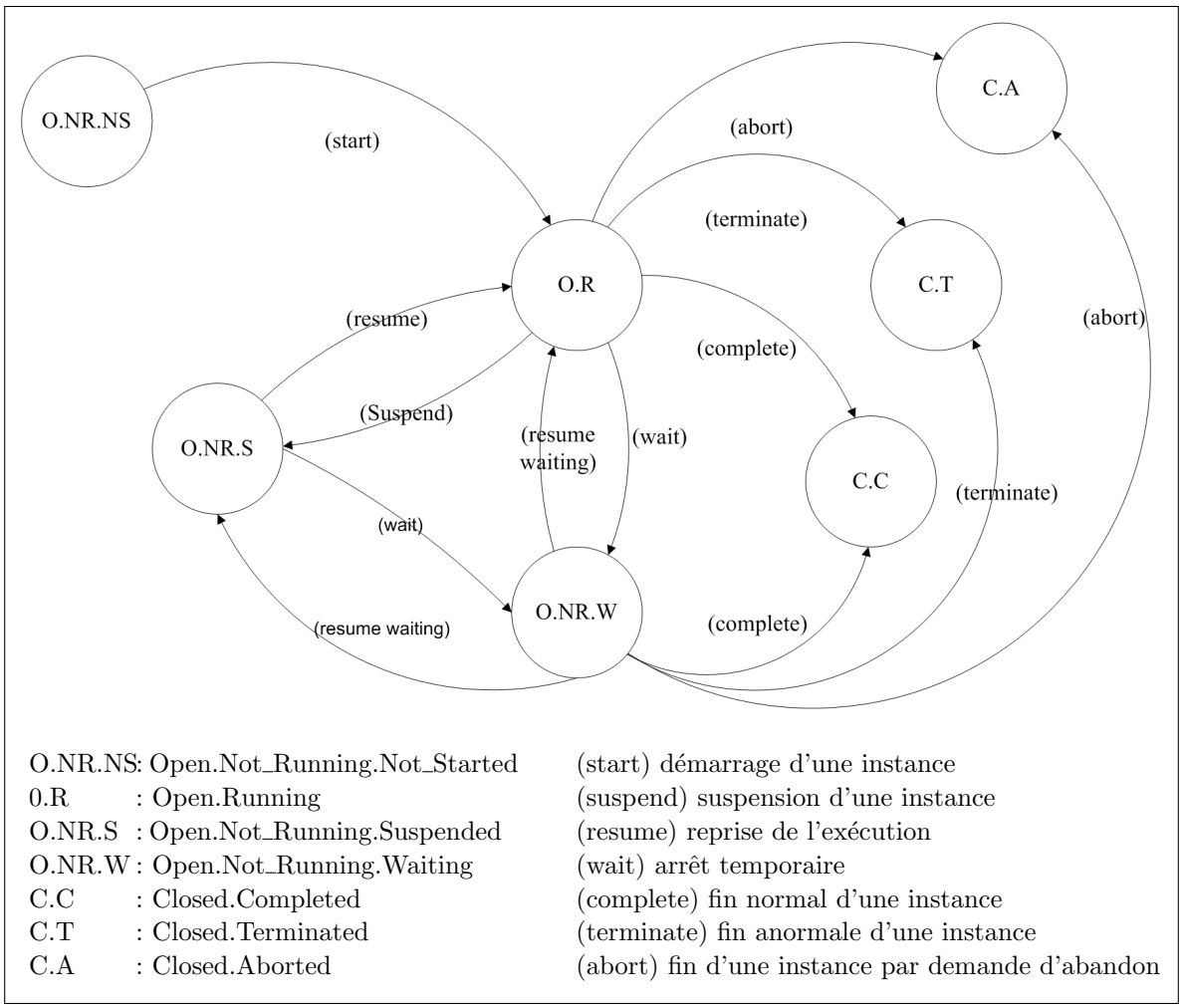


FIG. 5.1 – États des WfExecutionObject de COW

Nous allons maintenant voir plus en détail la mise en œuvre de la flexibilité dans COW. Nous commencerons par voir les états que peuvent avoir les instances de processus, d'activité et de tâche. Puis nous nous intéresserons à la gestion des instances de modèles de processus et d'activité et nous verrons les différentes possibilités d'adaptation des modèles que propose notre plate-forme. Enfin, nous verrons les aspects flexibilité liés au fonctionnement de la plate-forme elle-même, i.e., sa capacité à modifier son comportement, notamment par l'utilisation du patron stratégie.

5.2.1 États des WfExecutionObject

La figure 5.1 représente les différents états que peuvent prendre une instance de `WfExecutionObject` dans COW. Nous avons repris pour l'essentiel les états définis par le WMF en y ajoutant un état particulier appelé *Waiting*. Cet état correspond à la suspension temporaire du `WfExecutionObject` et de ses fils (e.g., une activité et les tâches associées). Nous utilisons cet état lorsque nous réalisons une modification dynamique du modèle, lorsque qu'une exception est détectée par le système et pour respecter les durées d'activités.

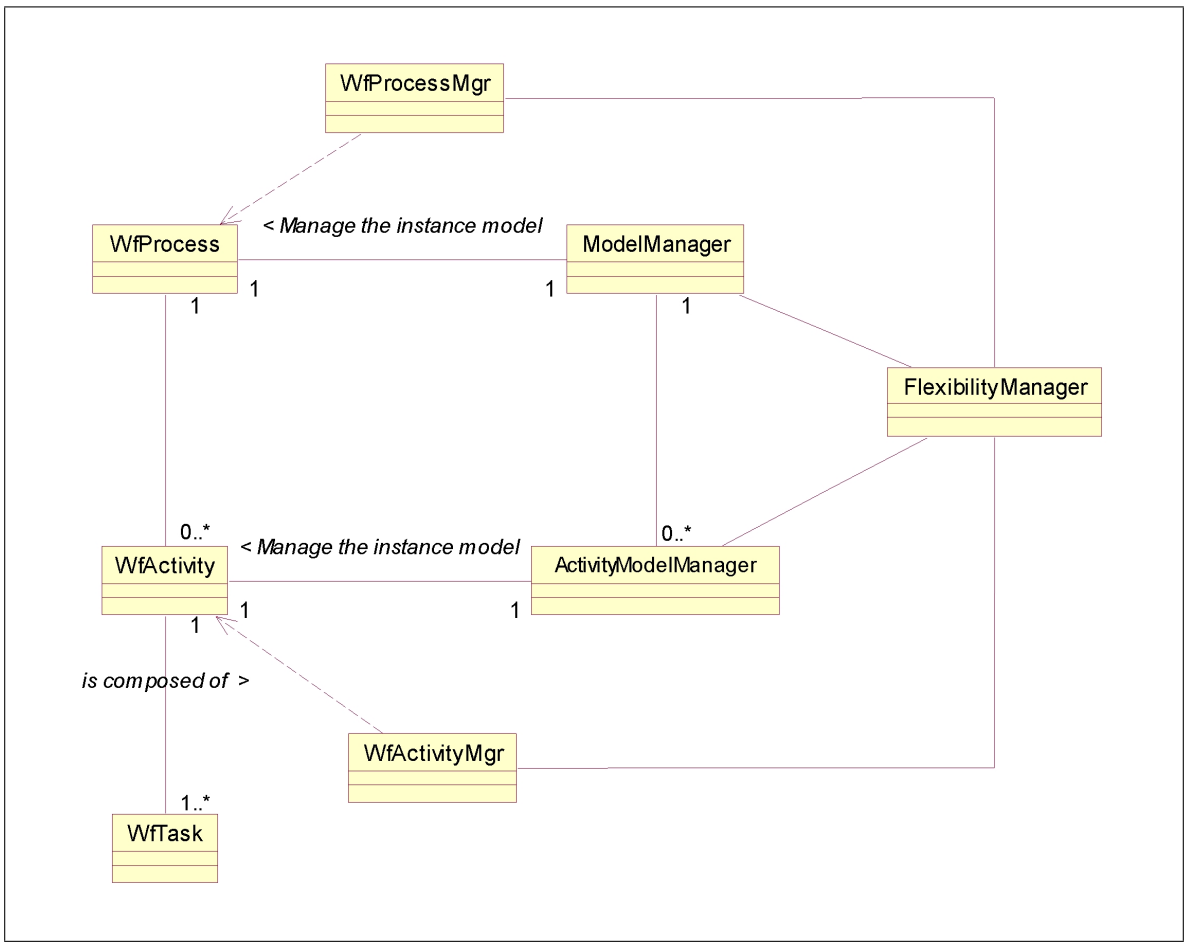


FIG. 5.2 – Gestion des modèles et de la flexiblité

5.2.2 Modification d'un modèle

La figure 5.2 montre les différentes classes qui interviennent dans le cadre de la flexibilité. Le composant `FlexibilityManager` est un EJB de type session qui se trouve au niveau des façades de notre moteur. Il offre une interface de haut niveau pour manipuler les instances de modèles de processus et d'activités. Son interface offre une méthode générique prenant en

paramètre un fichier XML décrivant les différentes modifications à apporter et il contient des méthodes plus spécifiques pour intervenir sur certains éléments du modèle (modification du temps, ...). Les objets `ModelManager` et `ActivityModelManager` offrent une méta-interface permettant l'ajout, le retrait ou la modification des différentes entités constituant le modèle. Ces méthodes sont utilisées notamment par le `FlexibilityManager`. La réalisation des transformations se déroule selon l'automate suivant (figure 5.3) :

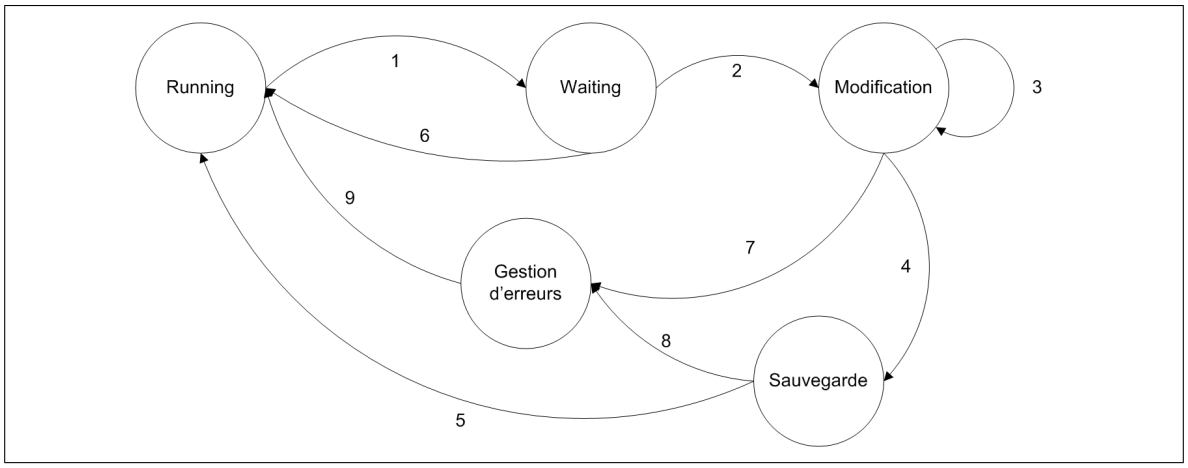


FIG. 5.3 – Automate de réalisation des transformations dynamiques de modèles.

1. Afin de réaliser une modification, le `WfExecutionObject` passe dans l'état `waiting` ;
2. Une fois cet état atteint, les modifications peuvent commencer.
3. Les modifications sont réalisées une à une.
4. Une fois l'ensemble des modifications effectuées, le nouveau modèle va être sauvegardé.
5. Si la sauvegarde se déroule correctement, le `WfExecutionObject` retourne dans l'état `Running`.

   L'occurence de problèmes lors de la transformation correspond aux transitions suivantes :

6. En cas de problème dans l'état `waiting`, l'objet retrouve son état initial.
7. Si une erreur survient lors des modifications, un mécanisme de rollback est déclenché afin de revenir dans l'état initial. Dans ce cas, toutes les modifications sont annulées car nous ne pouvons pas être certain que les changements effectués soient consistants.
8. De même, si la sauvegarde du nouveau modèle ne peut s'effectuer correctement, nous lançons le mécanime de rollback.
9. Après l'annulation de toutes les modifications, l'objet retrouve son état initial.

Pour faciliter la visualisation des modèles et leurs modifications, nous avons développé un outil de gestion et d'administration, visible à la figure 5.4. Il offre toutes les possibilités de changements que nous avons présentées précédemment (ajout, retrait, modification). Il contient également des fonctionnalités de surveillance des événements internes du moteur et permet la manipulation des différents modèles de processus et d'activités.

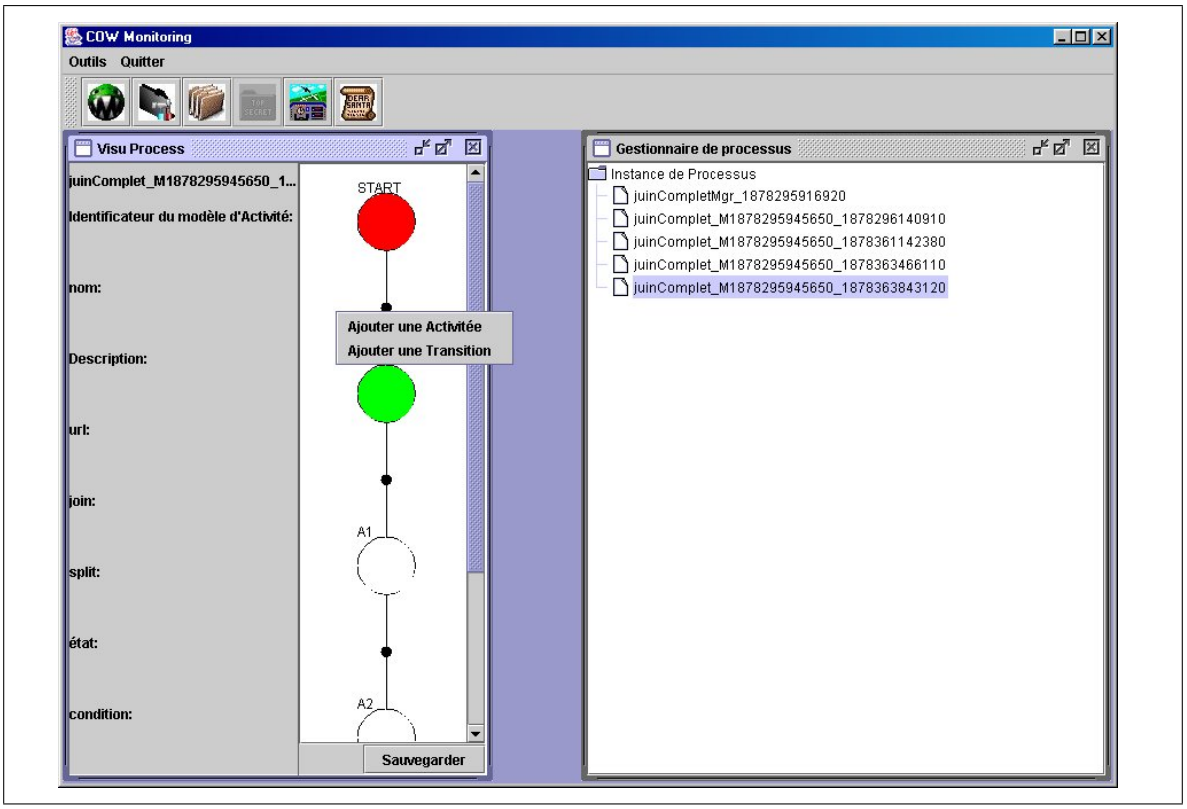


FIG. 5.4 – Outil d'administration de COW

5.2.2.1 Langage de modification

Afin d'exprimer les différentes modifications que nous souhaitons réaliser sur une instance, nous avons défini un langage de modifications. Il permet d'exprimer les différents besoins (ajout, retrait, modification) pour chaque élément de la structure d'exécution (processus, activité et tâche). Un exemple est visible à la figure 5.5. Dans cet exemple, nous allons ajouter, dans un modèle de processus, un activitylink qui aura pour identifiant `A34` et qui pointera sur le modèle d'activité `New_activity` ainsi qu'une transition sortant de cette activité. Nous allons modifier une transition pour la faire maintenant arriver vers l'activité que nous venons d'ajouter. Puis nous allons supprimer une activité et les deux transitions qui lui étaient attachées.

```
<InstanceModification>
    <ProcessId Id="12439" />
    <Add>
        <ActivityLink Id="A34" URL="New_activity" />
        <Transition Id="T8" From="A34" To="A23"/>
    </Add>
    <Modify>
        <TransitionModification>
            <TransitionId Id="T3"/>
            <Transition Id="T3" From="A22" To="A34">
        </TransitionModification>
    </Modify>
    <Remove>
        <ActivityId Id="A11"/>
        <TransitionId Id="T2">
        <TransitionId Id="T4">
    </Remove>
</InstanceModification>
```

FIG. 5.5 – Langage de modification des instances

5.3 La gestion du temps

Comme nous l'avons vu dans le chapitre 3, la gestion du temps est un élément important de la modélisation et de l'exécution d'un processus de Workflow. Dans le cas de l'éducation à distance, la prise en compte du temps permet de limiter l'accès aux ressources et aux tâches et évite dans le cadre d'une formation de groupe, une dérive trop importante entre les apprenants les plus rapides et les plus lents. L'avancement du groupe est ainsi plus homogène. Cette connaissance des contraintes temporelles permet également au tuteur et au référent de vérifier la charge de travail de l'apprenant et évite les excès (trop ou trop peu de travail). Chaque élément du modèle représentant un travail (Processus, activité, tâche) dispose de deux notions temporelles. La première, appelée durée, représente le temps minimal ou maximal pendant lequel l'élément devra se trouver dans l'état « running ». Le temps minimal ou maximal permet de ralentir les apprenants qui auraient tendance à passer trop vite à la suite du processus. Le temps maximal, au contraire, évite que l'apprenant ne s'attarde trop. La deuxième notion, appelée date limite, spécifie les périodes durant lesquelles le travail doit se situer.

Dans cette section, nous commencerons par montrer plus précisément la modélisation du temps, puis nous présenterons la mise en œuvre dans notre plate-forme.

5.3.1 Modélisation de l'aspect temporel

Lors de la modélisation d'un processus, d'une activité ou d'une tâche, il est possible de spécifier les différentes contraintes temporelles. Un exemple de spécification du temps est visible

à la figure 5.6. Comme nous l'avons expliqué dans la section précédente, elle se décompose en deux parties :

une partie durée (`limit`) qui peut être minimale (`MinTime`) ou maximale (`MaxTime`).

une partie date limite (`deadline`) qui peut être celle de départ (`StartDeadLine`) ou celle de fin (`EndDeadLine`).

Chacune de ces parties dispose d'un élément `Strategy` optionnel permettant de spécifier le comportement du moteur lors d'un dépassement de contrainte temporelle.

Il est possible d'exprimer les dates limites de deux manières possibles :

Date absolue. En absolu signifie que la date spécifiée correspond à une date de calendrier normal. Ce cas est relativement rare car il rend les modèles très spécifiques et ne permet pas de les réutiliser facilement. Nous utilisons pour cela l'élément `date`. Ce dernier comporte 6 attributs permettant d'exprimer l'année, le mois, le jour, l'heure, les minutes et les secondes. Dans l'exemple, l'activité (processus, tâche) devra être terminé pour le 11 décembre 2003 à 13h34 et 41 secondes.

Date relative. L'expression des dates en relatif est la méthode la plus utilisée. Elles sont par défaut calculées à partir de la date de création de l'activité (ou processus, ou tâche). Ces dates sont spécifiées par l'intermédiaire de l'élément `AdjustedTime`. Il contient également l'élément `date` mais pour lequel les attributs représentent une durée. Ainsi dans l'exemple, le temps minimal d'exécution devra être de 3 jours. Nous utilisons un élément `Threshold` optionnel pour spécifier un ajustement du décompte. Dans notre exemple, cela signifie que nous ne commencerons à décompter les 3 jours qu'à partir de 8 heure.

Ces contraintes temporelles peuvent être relâchées dans le cadre d'un enseignement de type formation continue, où l'apprenant, qui travaille généralement seul, peut avancer complètement à son rythme. Il est possible de modifier, retirer et ajouter des contraintes dynamiquement lors de l'exécution en passant par la façade de gestion de la flexibilité. Il faut cependant noter que si l'on reporte une date limite absolue, à l'heure actuelle nous ne répercutons pas de manière automatique ce décalage sur les autres dates limites absolues suivantes.

La stratégie de calcul par défaut pour les dates relatives est de prendre l'heure de création comme point de départ. Nous avons également une stratégie excluant les samedis et les dimanches du calcul. Cette stratégie s'applique plus particulièrement dans les entreprises où les apprenants n'accèdent au système que pendant les heures de travail.

```
<cowl:Limit>
    <cowl:MinTime>
        <cowl:TimeOption>
            <cowl:AdjustedTime>
                <cowl:Date day="3"/>
                <cowl:Threshold hour="8" minute="0" second="0"/>
                <cowl:Strategy> fr.noce.WeekEndStrategy </cowl:Strategy>
            </cowl:AdjustedTime>
        </cowl:TimeOption>
        <cowl:Strategy> fr.noce.Wait </cowl:Strategy>
    <cowl:/MinTime>
    <cowl:MaxTime>
        <cowl:TimeOption>
            <cowl:AdjustedTime>
                <cowl:Date day="8"/>
            </cowl:AdjustedTime>
        </cowl:TimeOption>
        <cowl:Strategy> fr.noce.AutomaticValidate </cowl:Strategy>
    </cowl:MaxTime>
</cowl:limit>
<cowl:deadline>
    <cowl:StartDeadLine>
        <cowl:TimeOption/>
            <cowl:AdjustedTime>
                <cowl:Date day="3"/>
                <cowl:Threshold hour="8" minute="0" second="0"/>
                <cowl:Strategy> fr.noce.WithoutWeekEnd </cowl:Strategy>
            </cowl:AdjustedTime>
        <cowl:Strategy> fr.noce.automaticStart </cowl:Strategy>
    </cowl:StartDeadLine>
    <cowl:EndDeadLine>
        <cowl:TimeOption>
            <cowl:Date year="2003" month="12" day="11"
                hour="13" minute="34" second="41"/>
        </cowl:TimeOption>
        <cowl:Strategy> fr.noce.sendMail </cowl:Strategy>
    </cowl:EndDeadLine>
</cowl:deadline>
```

FIG. 5.6 – Exemple de contraintes temporelles en COWL

5.3.2 Mise en œuvre

Pour la vérification des contraintes temporelles, nous utilisons les fonctions de timer fourni par JMX[10]. Cette technologie [JMX03] a pour objectif de fournir une interface standardisée pour la gestion et la surveillance d'applications.

Nous vérifions le temps minimum d'exécution lorsque l'activité (ou le processus ou la tâche) se termine. Si le temps d'exécution est inférieur à la durée minimale, nous plaçons l'activité (respectivement le processus, la tâche) dans l'état « waiting » en positionnant un indicateur de temps minimum non atteint. Toutes les X secondes (X étant configurable), nous vérifions toutes les instances dans l'état :

running pour qu'elles ne dépassent pas le temps maximum ni la date limite de fin ;

not started pour s'assurer qu'elles n'ont pas dépassé la date limite de départ ;

waiting et ayant l'indicateur de temps minimum. Si le temps minimal est atteint, nous passons l'activité dans l'état `completed`.

Lorsqu'une violation de contrainte est trouvée, nous appliquons la stratégie spécifiée s'il y en a une, sinon nous utilisons notre stratégie par défaut. Cette stratégie consiste à passer l'activité fautive dans l'état « waiting » et à envoyer un courrier électronique au responsable du module de formation et aux personnes devant réaliser l'activité. Nous avons également développé d'autres stratégies, comme la validation automatique du démarrage ou de l'arrêt.

5.4 Aspects utilisateur

Bien que les aspects utilisateur n'aient pas été au centre de nos préoccupations, nous nous sommes tout de même intéressés à la manière dont nous pouvions présenter les différentes informations relatives à notre système. Ces travaux furent réalisés en collaboration avec José Rouillard, maître de conférences au laboratoire TRIGONE [VR02]. Depuis le milieu des années 1990 et la tendance au client léger, l'approche la plus couramment mise en œuvre pour la construction des interfaces utilisateurs est l'utilisation des technologies du web et plus particulièrement le langage HTML. Cette solution permet notamment aux utilisateurs d'accéder à leurs applications depuis n'importe quelle machine disposant d'un navigateur internet offrant ainsi une certaine mobilité. Cependant, avec l'apparition de nouveaux périphériques nomades, cette approche doit être revue, car même si l'on trouve des navigateurs web sur les PDA[11], les caractéristiques d'affichage et d'interaction sont très différentes [Che02]. Dans le domaine de l'éducation à distance on voit ainsi apparaître la notion d'apprentissage mobile (*M-learning*). On assiste en parallèle à une volonté de personnaliser les informations présentées à l'utilisateur en fonction de son profil. Cette personnalisation peut, par exemple, se reposer sur le niveau de compétence et le parcours de formation de la personne. Nous n'abordons pas ici les problèmes relatifs à l'acquisition et à l'utilisation des profils utilisateurs.

[10] **J**ava **M**anagement E**x**tension
[11] **P**ersonal **D**igital **A**ssistant

L'objectif principal de ce travail est de montrer d'un point de vue technique comment nous pouvons supporter différents types de périphériques et de présenter la manière de personnaliser le contenu en provenance de notre moteur de workflows.

5.4.1 Mise en œuvre dans la plate-forme

5.4.1.1 Technologies existantes

Les trois principaux langages à balise permettant de réaliser des interfaces utilisateurs pour clients légers sont HTML, WML et VoiceXML. Ces langages offrent l'avantage d'être indépendants de la plate-forme d'exécution et ne nécessitent qu'une installation minimum de logiciels sur les clients. Nous ne reviendrons pas sur le premier langage qui est suffisamment popularisé. Le deuxième langage, WML (Wireless Markup Language), est utilisé par des navigateurs WAP (Wireless Application Protocol) [WAP03] comme des téléphones GSM par exemple. Le langage VoiceXML [Voi03] [BBC+01] supporte la description d'interaction d'un utilisateur utilisant un téléphone. Il permet notamment de spécifier des éléments textuels qui seront synthétisés et de traiter les demandes et les réponses des clients que cela soit par une reconnaissance de la parole ou par l'utilisation du clavier téléphonique.

5.4.1.2 Architecture

Pour réaliser facilement des interfaces utilisateurs adaptées à différents types de périphériques, nous nous sommes appuyés sur une architecture multi-niveaux et sur l'utilisation du couple XML/XSL comme support aux informations et à leurs transformations. L'architecture est présentée à la figure 5.7.

Le premier niveau de l'architecture est composé des différents périphériques utilisés pour l'accès à notre plate-forme. A l'heure actuelle, nous supportons :

Ordinateur personnel (PC/MAC). Ces périphériques offrent la plus grande possibilité d'interaction qui soit, à la fois textuelle, graphique et/ou sonore. Comme nous souhaitons des clients légers, les utilisateurs accèdent au système via un navigateur web.

Assistant Numérique Personnel (PDA). Ces périphériques ont des écrans de taille plus restreinte (environ 200*320 pixels) et de forme différente (aspect plus allongé qu'un écran de PC). Le contenu textuel et graphique doit être redimensionné en tenant compte de ces paramètres. Ils possédent généralement une entrée et une sortie audio. Ces systèmes utilisent des navigateurs web classiques mais restreints en fonctionnalités (pas d'accès au téléchargement de fichiers par exemple) ou des navigateurs WAP.

Téléphone GSM. Ces périphériques offrent deux modes d'interactions. Le premier est un navigateur WAP. Le second est l'utilisation de la voix et du clavier. Ils disposent d'un écran de petite taille (4 à 8 lignes).

Téléphone fixe. Les interactions sont limitées à la voix et au clavier (DTMF[12]).

[12]Dual Tone Multi Frequency

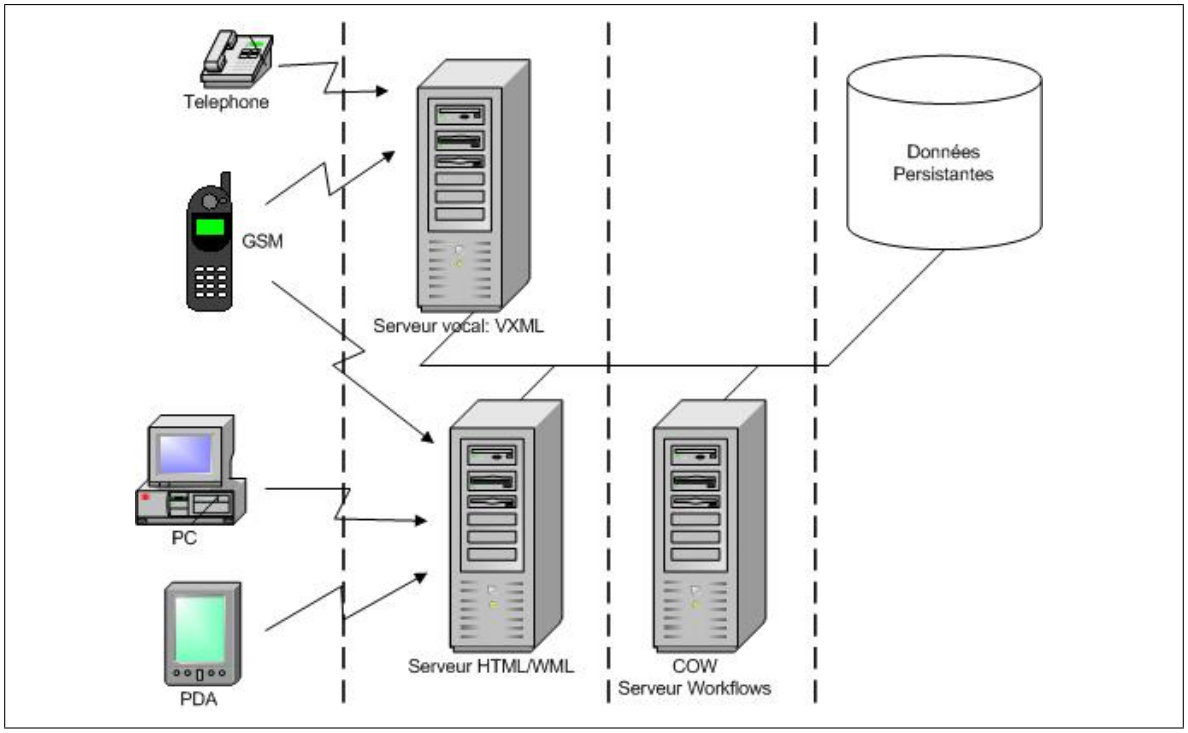


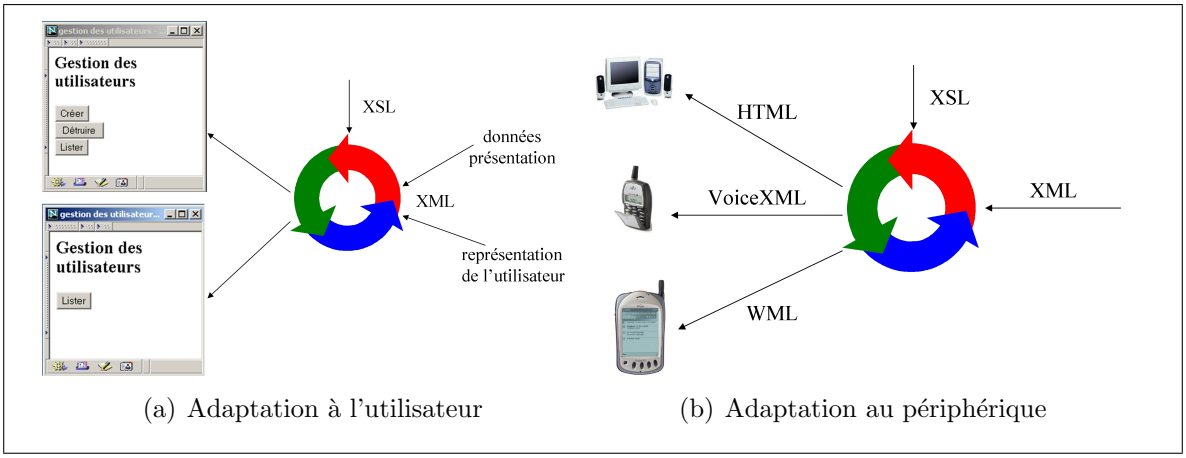
FIG. 5.7 – Architecture pour la présentation des données à l'utilisateur

Le deuxième niveau est composé des serveurs dédiés à la presentation/transformation des informations. On y trouve notamment un serveur vocal assurant la conversion VoiceXML en interaction vocale et la reconnaissance et compréhension des commandes passées vocalement par l'utilisateur. Un autre serveur est dédié aux transformations pour HTML et WML.

Le troisième niveau contient le système de workflows et le quatrième niveau assure la persistance des données.

5.4.1.3 Fonctionnement

Le fonctionnement global de la présentation des informations est illustré à la figure 5.8. Les serveurs dédiés à la présentation des informations interrogent les différentes façades du moteur de workflows. Ces dernières renvoient des données formatées en XML. La première étape consiste à adapter les informations au profil de l'utilisateur (rôle dans la plate-forme, compétences, préférences personelles). Nous avons choisi de réaliser cette transformation en premier car c'est elle qui nous permettra de filtrer un maximum les informations à présenter. De plus cela respecte le sens de circulation de l'information qui vient des serveurs contenant les profils et les données d'instances puis qui se dirige vers les périphériques. La deuxième étape consiste à accomoder les informations au périphérique utilisé, comme par exemple la taille des images, la mise en forme des textes, *etc.*



(a) Adaptation à l'utilisateur (b) Adaptation au périphérique

FIG. 5.8 – Transformation des interfaces utilisateur

5.4.2 Conclusion

Dans cette section, nous avons montré la mise en œuvre d'un accès multi-canal, i.e., le fait de communiquer à travers différents médium de communication (téléphone, SMS, PDA, ...), et personnalisable aux informations gérées par notre système de workflows. Le choix que nous avions fait pour COW de renvoyer les informations formatées en XML s'avère pertinent et en lien avec les travaux actuels dans le domaine de l'interaction homme-machine [Che02]. Ce travail nous a permis de valider ces choix. Les principales interrogations restantes concernent l'adaptation de la représentation des informations sur des périphériques à faibles capacités. Nous ne connaissons toujours pas quelles sont les informations pertinentes pour l'utilisateur et ceci en fonction du contexte d'interaction. Dans ce cadre là, nous devons encore mener un important travail sur les usages.

5.5 Conclusion du chapitre

L'objectif de ce chapitre était de présenter quelques détails d'implantation de notre moteur de workflows. Nous nous sommes intéressés dans un premier temps au support de la flexibilité de notre plate-forme selon deux aspects, la modification des modèles lors de l'exécution et le changement de comportement du moteur. Nous avons présenté notre architecture ainsi que le langage que nous avons défini pour la spécification des modifications. Nous avons ensuite présenté nos méthodes pour gérer les contraintes temporelles des processus, activités et tâches. Pour rappel, ces contraintes concernent la durée d'exécution et les dates limites. Nous avons exposé quelques une des stratégies permettant de calculer les durées ainsi que celles réagissant aux violations des contraintes. Nous avons terminé ce chapitre en introduisant les liens existants avec les recherches actuelles dans le domaine des Interactions Hommes-Machines et plus particulièrement dans le domaine de la communication multi-canal.

Chapitre 6

Environnement pédagogique sur COW

> People don't do serious work on Unix systems ; they send jokes around the world on USENET or write adventure games and research papers.
>
> – **E. Post**

Au cours des deux chapitres précédents, nous avons décrit le fonctionnement global et les différents mécanismes mis en œuvre pour supporter les différentes fonctionnalités (gestion de la flexibilité, du temps, ...). L'objectif du présent chapitre est de présenter une démarche de conception générique pour supporter l'exécution de scénarios pédagogiques par l'intermédiaire d'un moteur de workflows flexible, ainsi qu'un exemple de mise en œuvre. Nous verrons dans la première partie les différentes étapes de la démarche de conception en l'illustrant par l'utilisation du langage IMS-LD qui, comme nous l'avons vu dans le chapitre 2, répond le mieux à nos besoins actuels. Nous verrons de quelle manière nous pouvons réaliser les rapprochements entre une modélisation de scénarios pédagogiques et une modélisation de workflows. Ce rapprochement sera ensuite rafiné pour chacune des entités. Cela nous permettra notamment de montrer une manière de traiter les éléments qui ne sont pas présents simultanément dans les deux langages. Nous concluerons enfin cette première partie par la présentation d'une architecture de support d'un langage de modélisation pédagogique au dessus de COW. La partie suivante résumera une proposition d'interface homme-machine de présentation des informations aux apprenants, intégrable dans l'architecture précédente et s'appuyant sur les fonctionnalités de COW.

6.1 Langages pédagogiques

Dans le chapitre 2, nous avons présenté quelques langages de modélisation liés à la FOAD. L'intérêt de ces langages est de définir de manière complètement indépendante du système

CEN/ISSS	IMS-LD	COWL
Activity	Activity	Activity
Activity Structure	Activity Structure	Process
Objective	Learning objective	-
Prerequisite	Prerequisite	-
Outcome	Outcome	Data
Resource	Learning Object / Services	Data / Application
Environment	Environment	Data / Application
Role	Role	Participant
Person	Person	Participant
Dossier	-	-
Property	Property	-

TAB. 6.1 – Comparaison entre les entités de IMS-LD et COWL

d'exécution le parcours d'apprentissage, IMS-LD parlant même de « learning workflow ». Nous avons choisi de travailler à partir de ce dernier car il exprime de manière précise le déroulement des activités et cela de manière indépendante de l'approche pédagogique. De plus, ce langage pourrait vraissemblablement devenir un standard pour la description des parcours pédagogiques.

La première étape de la démarche de conception est le tissage de liens entre le méta-modèle de l'EML utilisé et celui du langage cible qui sera dans notre cas COWL. Cette étape est illustrée dans la section suivante.

6.1.1 Liens entre IMS-LD et COWL

Afin de rapprocher les deux modélisations, nous sommes passés par un « langage pivot » qui est celui du modèle de référence du CEN/ISSS. Le tableau 6.1 reprend toutes les entités du CEN/ISSS et de IMS-LD présentées au chapitre 2 ainsi que les entités de COWL. Nous ne reviendrons pas sur les correspondances entre IMS-LD et CEN/ISSS qui ont été présentées précédemment. Nous allons plutôt nous concentrer sur les relations existantes entre le méta-modèle du CEN et notre propre méta-modèle.

Le concept d'activité est similaire dans les deux cas. Le regroupement des activités se réalise en COWL par l'intermédiaire des processus et des transitions qu'ils contiennent. La notion d'objectif est complètement absente de notre modélisation. Elle peut uniquement se retrouver de manière implicite car la finalité d'un processus ou d'une activité peut être assimilée à certains objectifs. La production est représentée dans le workflow par les différentes données produites. Les ressources correspondent aux données utilisées ainsi qu'aux différentes applications utilisables pour réaliser l'activité. Ces éléments peuvent également se rapporter à la notion d'environnement d'exécution de l'activité. Les notions de rôle et de personne sont

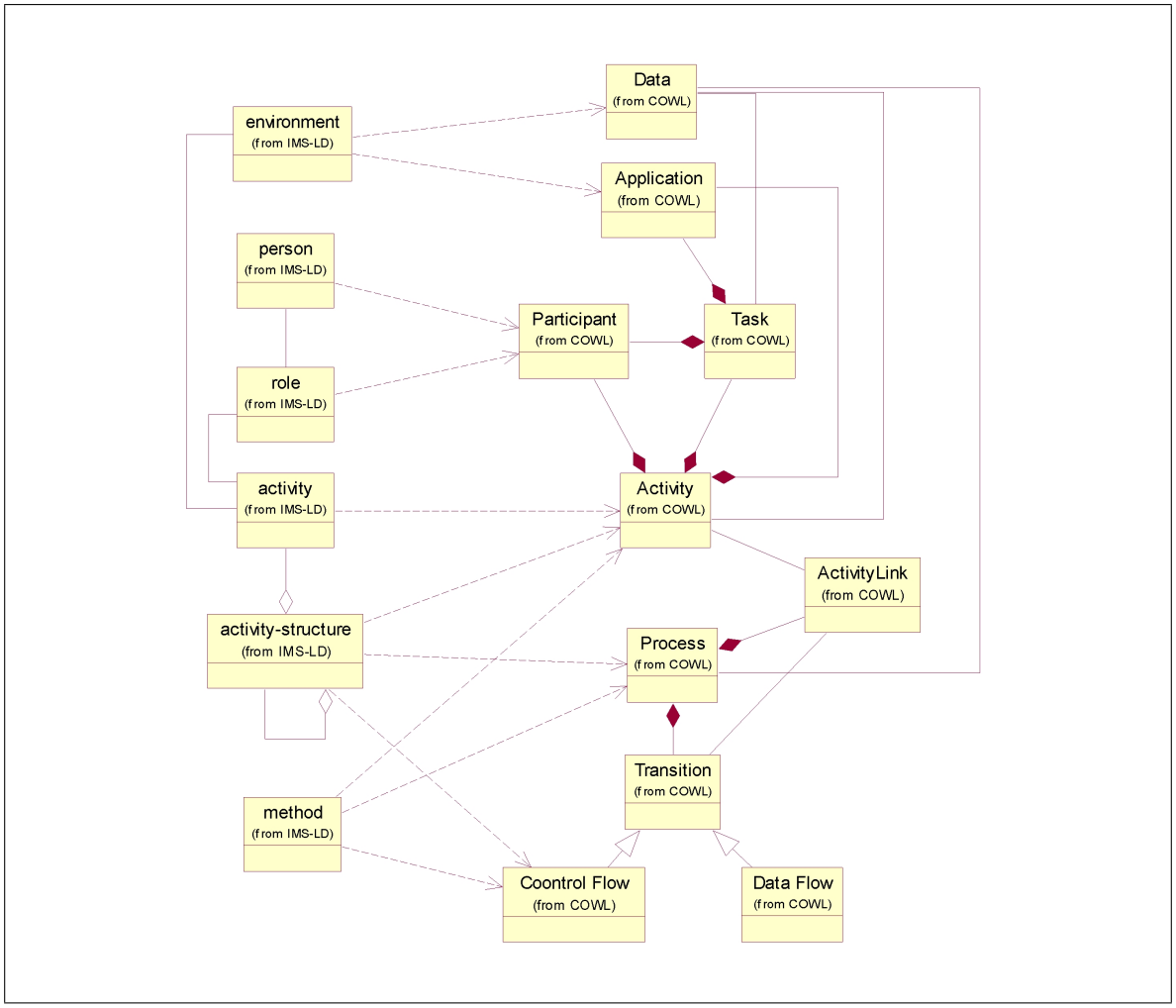


FIG. 6.1 – Liens entre IMS-LD et XPDL

contenues dans la notion de participant du workflow. Les concepts de dossier et de propriété ne sont pas traités par COW mais par les outils organisationels externes. Dans l'avenir cette comparaison reste intéressante lorsque nous souhaiterons utiliser un autre langage de modélisation pédagogique, la moitié de l'analyse sera déjà faite (celle pour COWL). Ce tableau nous permet directement de rapprocher IMS-LD et COWL en fournissant une vision à gros grain des liens possibles. Ces liens sont repris dans la figure 6.1

Nous allons maintenant voir plus en détail la composition des différentes entités des deux méta-modèles. Afin d'illustrer les ressemblances, les différences et les solutions envisageables, nous allons utiliser un exemple simple mais relativement complet provenant de la documentation de IMS-LD. Le lecteur intéressé pourra le retrouver dans l'annexe A. Le modèle décrit un enseignement dans le domaine de la maintenance aéronautique.

Afin de pouvoir exécuter un scénario pédagogique décrit en IMS-LD, nous devons dans un premier temps déterminer les éléments qui interviendront obligatoirement dans l'ordonnance-

ment des différentes activités, i.e., ceux qui sont nécessaires pour « alimenter » les différents constituants du moteur de workflows. Ces éléments seront différenciés de ceux qui n'interviennent pas de manière directe pour l'exécution mais qui gardent toute leur importance pour les enseignants et les apprenants, notamment les concepts pédagogiques. Nous cherchons à différencier les deux niveaux d'abstraction, celui utile pour l'exécution proprement dite et celui plus élevé adressant des aspects plus conceptuels.

Après un rapprochement rapide des différentes entités, la deuxième étape de la démarche de conception consiste à mettre en place plus finement le rapprochement des deux méta-modèles. Cette démarche est très spécifique aux deux langages utilisés. Pour étudier en détails les ressemblances et les différences des différentes entités, nous avons choisi de respecter l'ordre du tableau 6.1.

6.1.2 Le traitement des activités

La figure 6.2 représente une activité d'apprentissage exprimée en IMS-LD. Les éléments qui constituent l'activité de l'exemple sont :

- Un identifiant unique permettant de retrouver et référencer l'activité au sein du modèle complet. Dans notre exemple l'identifiant de l'activité est `LA-fuel-valve-lesson-intro`.
- Une description comprenant un titre (`Activity description title`) et un item. Cet item référence une ressource externe qui sera utilisée lors de l'instantiation de l'activité. Cette ressource est référencée via un identifiant unique (`RES-fuel-valve-lesson-intro`).

Les autres éléments non représentés sont les objectifs pédagogiques et le temps maximal d'exécution.

```
<imsld:learning-activity identifier="LA-fuel-valve-lesson-intro">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-fuel-valve-lesson-intro"
                    identifierref="RES-fuel-valve-lesson-intro">
            <imsld:title>Intro fuel valve</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
```

FIG. 6.2 – Description d'une activité en IMS-LD

La transformation de l'exemple en COWL est visible à la figure 6.3. Nous créons un modèle d'activité de même identifiant et de type *implementation*. La ressource décrite par la balise `item` de IMS-LD est considérée comme un des outils du moteur de worklows. Nous réutili-

sons le même identifiant mais nous ne faisons plus référence au contenu lui-même (attribut `identifierref`). Ce dernier est déplacé dans notre modèle d'instance.

```
<cowl:activity Id="LA-fuel-valve-lesson-intro">
    <cowl:description> </cowl:description>
    <cowl:Implementation>
        <cowl:tool Id="I-fuel-valve-lesson-intro">
            <cowl:description>Intro fuel valve</cowl:description>
        </cowl:tool>
    </cowl:Implementation>
</cowl:activity>
```

FIG. 6.3 – Description d'une activité en COWL

6.1.3 Structuration des activités

Un exemple de la structuration des activités est visible à la figure 6.4. Nous pouvons voir les deux types de structuration : la séquence et la sélection. Dans le cas de la séquence chaque activité doit être faite dans l'ordre indiqué. Dans notre exemple, l'activité `LA-fuel-valve-lesson-intro` sera réalisée en premier et sera suivie de l'activité `LA-fuel-valve-lesson-theory`. Dans le cas d'une sélection (`structure-type=selection`), l'apprenant devra réaliser un certain nombre de tâches. Ce nombre est spécifié dans l'attribut `number-to-select`. Dans l'exemple présenté, l'apprenant devra réaliser les deux activités mais il pourra les traiter dans l'ordre qu'il veut.

```
<imsld:activity-structure identifier="AS-introduction"
                          number-to-select="2"
                          structure-type="sequence">
    <imsld:title/>
    <imsld:learning-activity-ref ref="LA-fuel-valve-lesson-intro"/>
    <imsld:learning-activity-ref ref="LA-fuel-valve-theory"/>
</imsld:activity-structure>
<imsld:activity-structure identifier="AS-fuel-valve-lessons"
                          number-to-select="2"
                          structure-type="selection">
    <imsld:title/>
    <imsld:learning-activity-ref ref="LA-lesson-hazards"/>
    <imsld:learning-activity-ref ref="LA-lesson-components"/>
</imsld:activity-structure>
```

FIG. 6.4 – Structuration des activités en IMS-LD

Dans le cadre de COW, nous utilisons la notion de processus pour ordonner les différentes activités (figure 6.5). Un ordonnancement séquentiel est réalisé par l'intermédiaire des tran-

sitions (transition `T1` dans notre exemple pour ordonner les deux premières activités). Pour réaliser une sélection, nous utilisons des activités de routage afin de lancer en parallèle les différentes activités et les synchroniser via les préconditions de l'activité de routage d'arrivée.

```
<cowl:process>
    <cowl:id> </cowl:id>
    <cowl:activities>
        <cowl:activityLink id=A1>
            <cowl:url> LA-fuel-valve-lesson-intro </cowl:url>
        </cowl:activityLink>
        <cowl:activityLink id=A2>
            <cowl:url> LA-fuel-valve-theory </cowl:url>
        </cowl:activityLink>
        <cowl:activityLink id=AR1>
            <cow:url> Route_Start_Selection </cowl:url>
        </cowl:activityLink>
        <cowl:activityLink id=A3>
            <cowl:url> LA-lesson-hazards </cowl:url>
        </cowl:activityLink>
        <cowl:activityLink id=A4>
            <cowl:url> LA-lesson-components </cowl:url>
        </cowl:activityLink>
        <cowl:activityLink>
            <cow:url> Route_End_Selection </cowl:url>
            <cowl:TransitionRestriction>
                <cowl:Join type="AND">
                <cowl:in_transition_true>
                        2
                </cowl:in_transition_true>
            </cowl:join>
            </cowl:TransitionRestriction>
        </cowl:activityLink>
    </cowl:activities>
    <cowl:transitions>
        <cowl:transition id=T1 From=A1 To=A2/>
        <cowl:transition id=T2 From=A2 To=AR1/>
        <cowl:transition id=T3 From=AR1 To=A3/>
        <cowl:transition id=T4 From=AR1 To=A4/>
        <cowl:transition id=T5 From=A3 To=AR2/>
        <cowl:transition id=T6 From=A4 To=AR2/>
    </cowl:transitions>
</cowl:process>
```

FIG. 6.5 – Structuration des activités en COWL

6.1.3.1 Ordonnancement des activités

Il existe dans IMS-LD une différence entre la structuration des activités, i.e. leur regroupement dans des blocs de plus haut niveau et le séquencement des différentes activités, i.e., leur ordonnancement. La structuration est réalisée par l'intermédiaire des entités `activity-structure` (paragraphe précédent). L'ordonnancement est quant à lui laissé à l'entité `method` (figure 6.6).

```
<imsld:method>
    <imsld:play identifier="PLAY-Boeing-simplified" isvisible="true">
        <imsld:act identifier="ACT-individualized-learning">
            <imsld:role-part identifier="RP-individualized-learning">
                <imsld:role-ref ref="R-learner"/>
                <imsld:activity-structure-ref ref="AS-boeing-simplified"/>
            </imsld:role-part>
            <imsld:complete-act>
                <imsld:when-role-part-completed ref="RP-individualized-learning"/>
            </imsld:complete-act>
        </imsld:act>
        <imsld:complete-play>
            <imsld:when-last-act-completed/>
        </imsld:complete-play>
    </imsld:play>
</imsld:method>
```

FIG. 6.6 – Ordonnancement des activités en IMS-LD

Cette distinction n'existe pas dans la modélisation de COW. Nous utilisons pour la structuration et l'ordonnancement la notion de processus. Dans l'exemple, la transformation de l'élément `method` correspondra à la création d'un modèle de processus, similaire à celui de la figure 6.5, comportant deux *activityLinks*, un d'identifiant START pointant sur une activité de type sous-processus correspondant à la structure `AS-boeing-simplified`, et un d'identifiant END pointant sur une activité de type routage.

6.1.4 Traitement des objectifs

Les objectifs pédagogiques de IMS-LD sont exprimés par l'intermédiaire de la balise `learning-objectives` (figure 6.7). Ils peuvent être situés au niveau du module d'enseignement et dans chaque activité.

Cette notion est complètement absente de COWL. Pour la prendre en compte, nous utilisons les éléments `ExtendedAttribute`, car ils n'ont aucune influence sur le comportement du moteur. Mais ils doivent pouvoir être retrouvés facilement pour être présentés à l'utilisateur (figure 6.8). Ce dernier point sera explicité dans la section 6.2.1.

```
<imsld:learning-objectives>
    <imsld:title>Learning objectives title</imsld:title>
    <imsld:item identifier="LOB-learning-objectives"
            identifierref="RES-learning-objectives">
        <imsld:title>Learning objective title</imsld:title>
    </imsld:item>
</imsld:learning-objectives>
```

FIG. 6.7 – Objectifs en IMS-LD

```
<cowl:ExtendedAttributes Name="Learning_Objectives">
    <imsld:title>Learning objectives title</imsld:title>
    <imsld:item identifier="LOB-learning-objectives"
            identifierref="RES-learning-objectives">
        <imsld:title>Learning objective title</imsld:title>
    </imsld:item>
</cowl:ExtendedAttributes>
```

FIG. 6.8 – Objectifs en COWL

6.1.5 Traitement des prérequis

La notion de prérequis permet d'indiquer les connaissances minimums pour la réalisation de l'activité (figure 6.9).

```
<imsld:prerequisites>
    <imsld:title>Prerequisites title</imsld:title>
    <imsld:item identifier="PREQ-prerequisites" identifierref="RES-prerequisites">
        <imsld:title>Prerequisite title</imsld:title>
    </imsld:item>
</imsld:prerequisites>
```

FIG. 6.9 – Prérequis en IMS-LD

Nous les traitons dans COW de la même manière que les objectifs en utilisant les éléments `ExtendedAttribute`. La transformation des prérequis est présentée à la figure 6.10.

```
<cowl:ExtendedAttribute Name="prerequisite">
    <imsld:title>Learning objectives title</imsld:title>
    <imsld:item identifier="LOB-learning-objectives"
            identifierref="RES-learning-objectives">
        <imsld:title>Learning objective title</imsld:title>
    </imsld:item>
</cowl:ExtendedAttribute>
```

FIG. 6.10 – Prérequis en COWL

6.1.6 Traitement de l'environnement

L'environnement de IMS-LD contient l'ensemble des informations et des outils qui forment le cadre des différentes activités (figure 6.11). Il contient au minimum une ressource pédagogique *learning-object*. Chaque *item* représente un lien vers une ressource. L'environnement peut également contenir différents services, comme par exemple un service permettant d'envoyer des mails aux utilisateurs possédant un certain rôle.

```
<imsld:environments>
    <imsld:environment identifier="E-interactive-electronic-training-manual">
        <imsld:title>Interactive Electronic Technical Manual</imsld:title>
        <imsld:learning-object identifier="Training-manual-LO">
            <imsld:item  identifier="Training-manual-item"
                identifierref="Training-res"/>
            <imsld:service identifier="Mail" isvisible="true">
                <imsld:send-mail select="all-persons-in-role">
                    <imsld:email-data>
                        <imsld:role-ref ref="R-learner"/>
                    </imsld:email-data>
                </imsld:send-mail>
            </imsld:service>
        </imsld:learning-object>
    </imsld:environment>
</imsld:environments>
```

FIG. 6.11 – Description d'un environnement en IMS-LD

Dans COWL, l'environnement d'exécution est représenté par les différentes applications offertes à l'utilisateur (figure 6.12). Pour la ressource pédagogique, nous reprenons uniquement l'identifiant et la référence de la ressource. Pour le service, nous devons en plus gérer un paramètre d'entrée représentant le rôle à prendre en compte. Nous avons défini dans notre architecture une correspondance entre les différents services offerts par IMS-LD et les applications réellements utilisées.

```
<cowl:application Id="Training-manual-item" ref="Training-res">
<cowl:application Id="Mail" ref="send-mail">
    <cowl:formalParameter IOMode="IN">
        <cowl:dataField Id="role">
            <cowl:initialValue> R-Learner </cowl:initialValue>
        </cowl:dataField>
    </cowl:formalParameter>
</cowl:application>
```

FIG. 6.12 – Description d'un environnement en IMS-LD

6.1.7 Le traitement des participants (rôle et personne)

Comme nous avons pu le montrer dans le chapitre 2, dans IMS-LD les participants sont désignés par le concept de rôle. Ce concept est décomposé en deux catégories ; *learner* qui représente les apprenants et *staff* qui représente les enseignants ainsi que tous les autres intervenants (ingénieur pédagogique, administrateur, *etc*). Chacun de ces deux rôles peut être subdivisé. La figure 6.13 représente la description du rôle utilisé dans notre exemple. Nous avons un seul rôle de type *learner* et d'identifiant `R-learner`.

```
<imsld:roles>
    <imsld:learner identifier="R-learner"/>
</imsld:roles>
```

FIG. 6.13 – Description d'un rôle en IMS-LD

En COWL, un participant est décrit par l'entité *participant*. En reprenant l'exemple précédent, le même rôle décrit en COWL est visible à la figure 6.14.

```
<cowl:participant Id="R-learner">
    <cowl:participantType Type="ROLE"/>
</cowl:participant>
```

FIG. 6.14 – Description d'un rôle en COWL

Bien que cela ne soit pas flagrant dans notre exemple, le pouvoir d'expression d'un participant dans COW est plus important que dans IMS-LD. Nous ne nous limitons pas à la notion de rôle, un participant peut même être une application comme par exemple un logiciel de correction automatique de QCM. Cela s'explique en partie par la conception générique de COW. Nous ne voulions pas le limiter à un champ d'applications particulier.

6.2 Du langage métier à son exécution

Pour permettre l'exécution d'un langage de modélisation pédagogique dans notre système, nous avons deux possiblités :

Transformation de modèle. Il s'agit de transformer les concepts d'un langage à un autre. Une solution peut être de tisser des liens entre les méta-modèles. Comme la majorité des EMLs sont décrits en XML et comme notre langage est également décrit en XML, il serait possible par l'intermédiaire de XSLT de réaliser les transformations. Cette transformation peux entrainer des pertes d'informations car il n'existe pas obligatoirement une relation 1-1 entre les concepts de l'EML et de notre système.

Encapsulation du modèle. Dans le cas d'encapsulation du modèle cela revient, en plus des transformeurs, à ajouter des composants qui serviront d'interfaces entre le moteur et le modèle originel. Ces composants se situent au niveau de la facade dans notre architecture en couche. Cette solution demande plus de développement mais évite la perte d'information et permet même un enrichissement de l'EML en lui fournissant des informations qui existent dans notre système.

6.2.1 Architecture

La figure 6.15 représente l'architecture mise en œuvre afin de permettre l'exécution de scénarios pédagogiques décrit en IMS-LD sur notre plate-forme. Celle-ci est composée des éléments suivants :

Le noyau fondamental de la façade. Ce noyau est composé de quatre parties distincte : deux convertisseurs statiques IMS-LD vers COWL et COWL vers IMS-LD, un module de sauvegarde des informations et un module de gestion des instances en cours d'exécution.

- **Un convertisseur IMS-LD vers COWL** permettant de transformer un modèle exprimé en IMS-LD en un modèle exprimé en COWL.
- **Un convertisseur COWL vers IMS-LD** permettant de transformer un modèle exprimé en COWL en un modèle exprimé en IMS-LD. Ce convertisseur ne peut fonctionner complètement que si les modèles COWL utilisés proviennent d'une prédédente transformation de IMS-LD vers COWL car certains éléments tels que les *objectifs* n'existent pas par défaut dans le langage COWL.
- **Un référentiel des modèles** qui sauvegarde les différents éléments en provenance de IMS-LD qui ne sont pas fondamentaux pour l'exécution du scénario pédagogique par COW mais qui sont importants dans un contexte pédagogique comme par exemple les objectifs à atteindre lors de la réalisation d'une activité.
- **Un gestionnaire d'instances** permettant de fournir au module interface utilisateur les différentes informations à présenter aux utilisateurs du service.

Un module d'interface utilisateur qui interroge le noyau fondamental afin de récupérer les différentes informations et qui ensuite les adapte pour les présenter aux différents utilisateurs en fonction de leurs rôles et de leurs périphériques (PC, PDA, ...).

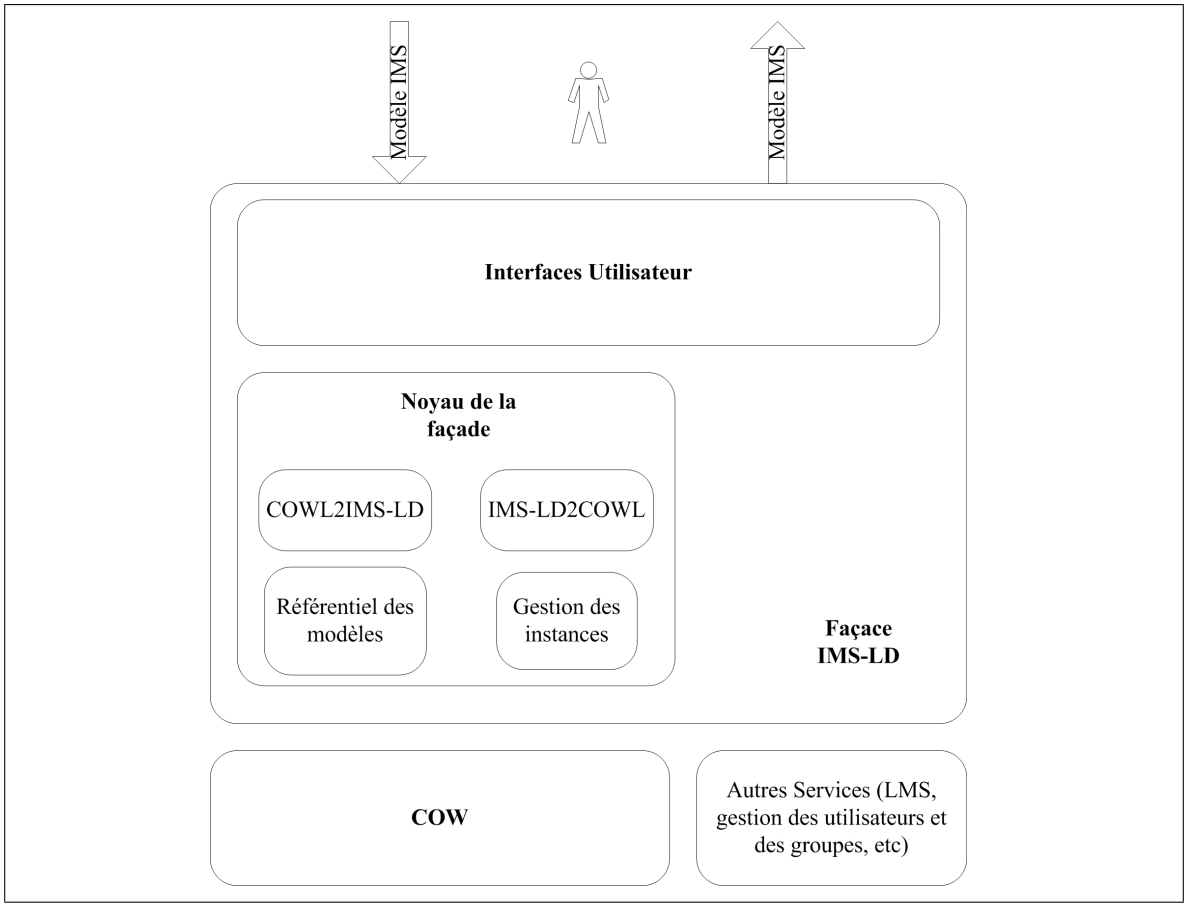


FIG. 6.15 – Architecture supportant IMS-LD dans COW

La façade s'appuie sur les services de COW et ainsi que sur d'autres services fournis par des plates-formes existantes comme par exemple un service de gestion des utilisateurs et des groupes de PLACE [VP02].

6.3 Le système GAIN

Afin de montrer un cas d'utilisation de notre plate-forme dans le cadre d'une formation à distance, nous avons mis en œuvre un guide d'apprentissage dans le cadre du système GAIN (*Guide d'Apprentissage Interactif Numérique*) [Rag03] [Goi03]. L'interface d'un tel système contient deux environnements différents :

- Un environnement dédié à l'apprenant où celui-ci trouvera la liste des différentes tâches qui lui sont affectées ainsi que son état d'avancement par rapport au temps alloué mais également par rapport à l'avancement global du groupe auquel il appartient ;
- Un environnement dédié aux tuteurs et aux référents où ils pourront trouver un ensemble d'outils permettant de visualiser la progression individuelle ou de groupe des

différents apprenants dont ils ont la charge. Ils disposeront également d'outils permettant la manipulation des différents modèles instantiés afin de pouvoir les visualiser et éventuellement les modifier pour répondre à des besoins circonstanciés.

6.3.1 Mise en œuvre

6.3.1.1 Interfaces de l'apprenant

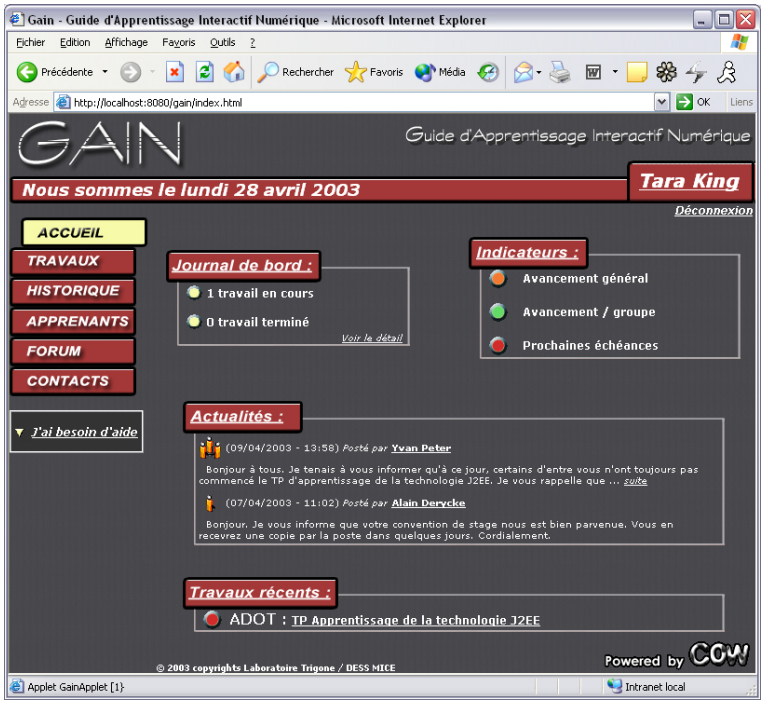


FIG. 6.16 – Page d'accueil de GAIN

La page d'accueil de l'apprenant est représentée à la figure 6.16. Elle comporte les éléments suivants :

- un **journal de bord** contenant le nombre de travaux en cours et de travaux terminés. Cet élément permet à l'utilisateur de connaître rapidement s'il est à jour dans son travail.
- une série d'**indicateurs** permettant à l'utilisateur de situer son avancement par rapport à l'ensemble des tâches qu'il a à réaliser, par rapport aux autres membres de son groupe et par rapport aux différentes dates limites.
- une série de messages d'**actualités** adressés au groupe ou personnellement à l'étudiant.
- une liste des **travaux récents**, i.e., les différents modules dans lesquels il a réalisé des actions.

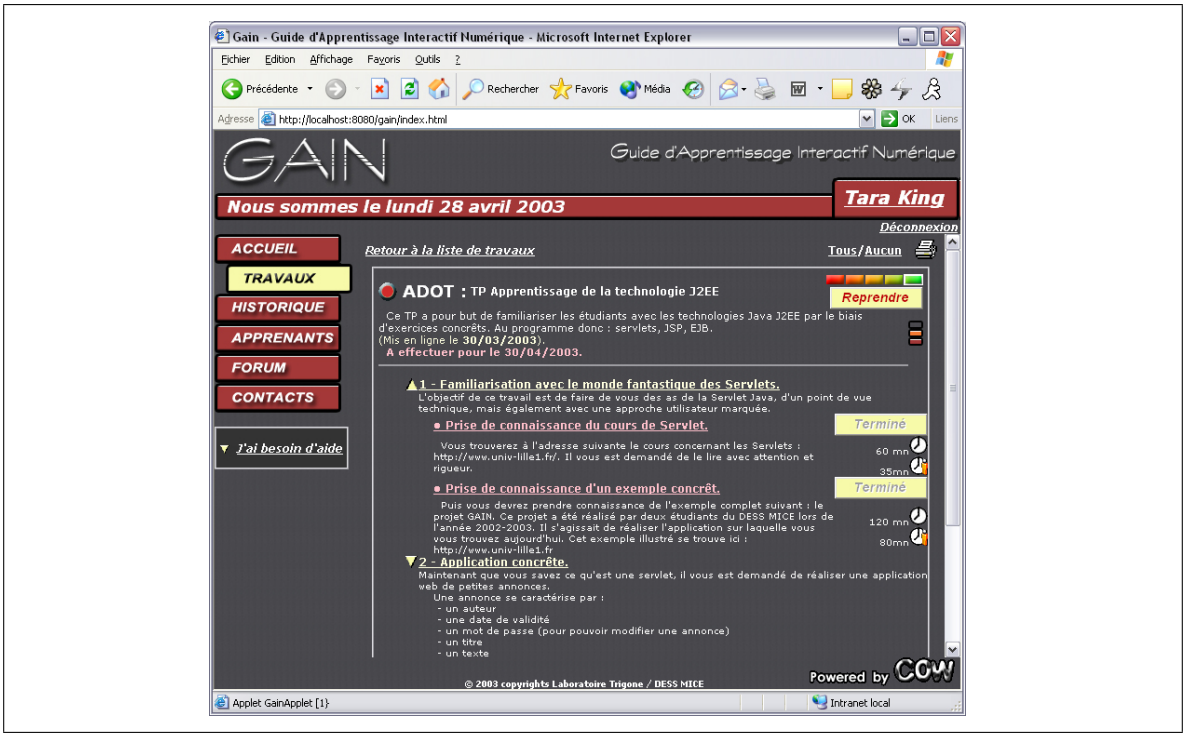


FIG. 6.17 – Contenu d'un module d'enseignement présenté par GAIN

Lorsqu'il accède à un module, l'étudiant se trouve face à la page présentée à la figure 6.17. Elle comporte les éléments suivants :

- Un en-tête comprenant l'intitulé exact du cours qui est dans notre exemple *ADOT : TP Apprentissage de la technologie J2EE*, un descriptif des objectifs, la date limite pour l'exécution et l'état d'avancement.
- Le découpage du module en différentes sections. Cette division peut se faire à autant de niveaux que l'on souhaite en profondeur. Pour chaque tâche il y a un descriptif, le temps estimé pour la réalisation, le temps effectif de l'étudiant et une opération (démarrer, suspendre, arrêter).

La figure 6.18 explique les différents constituants de l'interface d'un module.

6.3.1.2 Caractéristiques spécifiques

Afin de supporter ce guide d'apprentissage dans COW, certaines modifications sont nécessaires pour prendre en compte les caractéristiques spécifiques. De même que pour l'utilisation de modèles exprimés en IMS-LD, il a été nécessaire d'ajouter des informations dans notre méta-modèle. L'élément principal ajouté au métamodèle est la notion de temps estimé (*estimated time*) permettant à un enseignant ou à un ingénieur pédagogique d'indiquer le temps que devrait normalement durer une activité ou un module d'apprentissage. Cela permet à l'étudiant, lorsqu'il n'existe pas de contrainte temporelle, de pouvoir évaluer s'il a travaillé

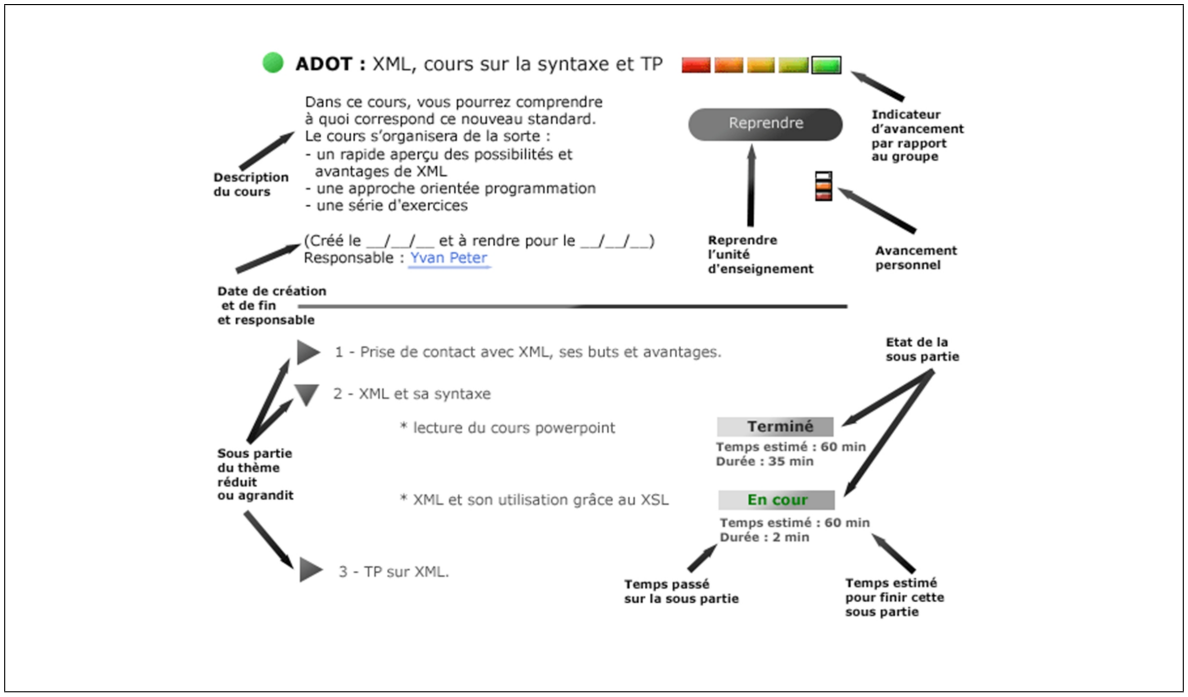


FIG. 6.18 – Explication du contenu d'un module

dans un temps « normal » ou s'il a été trop lent ou trop rapide. Les processus, activités et tâches de COW ont été augmentés de cette notion par l'utilisation des `extended attribute`.

6.3.1.3 Architecture

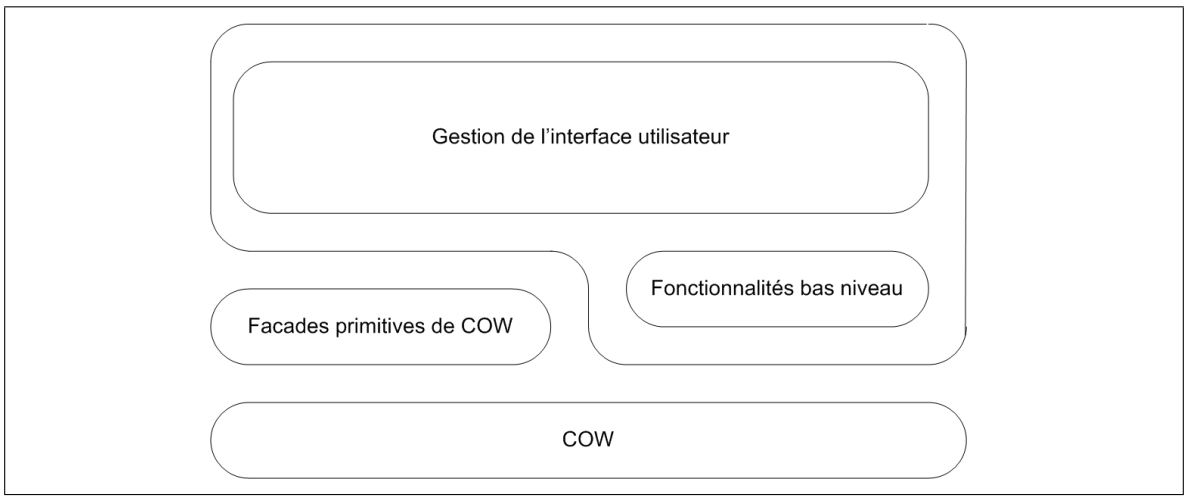


FIG. 6.19 – Architecture et positionnement de GAIN avec COW

L'architecture de GAIN est représentée à la figure 6.19. Elle s'appuie sur deux composants principaux, un composant de bas niveau communiquant directement avec le moteur de workflows et un composant dédié à la gestion de l'interface utilisateur et qui s'appuie sur le composant précédents ainsi que sur les différentes façades de COW.

Le composant de bas niveau peut être considéré comme une des façades de COW. Il contient principalement deux catégories de méthodes, celles gérant la notion de temps estimé et celles de recherche et de représentation de l'arbre des processus.

De plus l'infrastructure réalisée pour le guide d'apprentissage interactif s'intégre dans l'infrastructure gérant les interfaces utilisateurs (section 5.4).

6.4 Conclusion du chapitre

Dans ce chapitre, nous avons indiqué une démarche de conception permettant le passage d'un EML vers une plate-forme d'exécution basée sur un moteur de workflows. Cette démarche se décompose en trois étapes :

1. Détermination à gros grain des liens entre les deux méta-modèles. Cette partie est réalisée en utilisant le méta-modèle du CEN/ISSS comme pivot.
2. Raffinement des liens en déterminant les parties nécessaires à l'exécution et celles nécessaires au contexte pédagogique.
3. Développement des composants spécifiques à l'EML, les convertisseurs notamment, et leurs intégration dans l'architecture générique proposée.

Nous avons ensuite montré une interface utilisateur possible pour qu'un apprenant puisse visualiser et réaliser les différentes activités pédagogiques.

Ces différents travaux nous ont permis d'obtenir un retour d'expérience sur les choix de conception que nous avons réalisés. La possibilité d'extension de notre méta-modèle présentée au chapitre 4 nous a permis de prendre en compte efficacement certains éléments d'IMS-LD qui n'étaient pas essentiels pour l'exécution du modèle mais qui sont très importants pour l'apprenant comme par exemple la notion d'objectif et de prérequis. Cette extension fut également mise en œuvre dans le cadre du développement de GAIN pour la prise en compte de la notion de temps estimé. Dans le cadre de GAIN, nous avons également pu nous rendre compte que les interfaces du micronoyau et des différentes façades de bases que nous avions développés répondent bien aux différents besoins pour la constitution d'interfaces spécifiques à l'éducation à distance.

Troisième partie

Conclusion et perspectives

Chapitre 7

Conclusion et perspectives

« [...]Tout ce qu'on en sait est que la Réponse est Quarante-deux, ce qui ne clarifie pas le problème. »

Douglas Adam, ***La Vie, l'Univers et le Reste***.

Ce travail se situe dans le contexte de la Formation Ouverte et A Distance (FOAD) et vise à concevoir et réaliser une plate-forme pour l'exécution de scénarios pédagogiques. Cela nous a conduit à étudier les langages de modélisation pédagogiques ou Educational Modelling Language (EML). Nous avons défini trois catégories d'EML : les langages d'évaluation, les langages de structuration de resources et les langages de structuration des activités. Nous nous sommes plus particulièrement intéressé à la dernière catégorie et avons choisi IMS-LD afin de valider notre démarche. On peut remarquer que les langages de structuration d'activités présentent des points communs avec la gestion des processus en général, ce qui nous a amené à envisager l'utilisation d'un moteur de Workflows pour l'exécution des scénarios pédagogiques. Toutefois les moteurs existants ne répondent pas complètement aux contraintes de la FOAD. Nous avons donc choisi d'implémenter le moteur de Workflows COW (pour Cooperative Open Workflow) avec les objectifs suivants :

- Supporter différents types d'enseignements : individuels et de groupes y compris au sein d'un même processus ;
- Supporter les activités collaboratives, c'est-à-dire permettre à des groupes d'utilisateurs d'intervenir dans une activité avec des rôles et des tâches différentes ;
- Supporter une redéfinition dynamique du processus d'apprentissage afin de pouvoir l'adapter aux étudiants et à l'environnement ;
- Favoriser la réutilisation des modèles de processus et d'activités y compris après une modification d'un modèle existant.

Pour cela, nous avons choisi de nous placer dans le cadre des standards existants : ceux du Workflow Management Coalition (WfMC) en particulier en ce qui concerne XML Process

Definition Language (XPDL), le langage de description de processus ainsi que le Workflow Management Facility (WMF) de l'Object Management Group (OMG) dont nous avons réalisé une implantation. Pour cela, nous avons tiré partie de la technologie Enterprise JavaBeans et avons suivi une approche de type micronoyau permettant d'étendre facilement les fonctions du moteur.

Pour supporter la flexibilité et la possibilité de modifier les processus d'apprentissage, nous avons combiné une approche par méta-modèle afin d'assurer la réflexivité et l'intercession ainsi que l'approche par points ouverts afin de modifier le comportement du moteur. Le méta-modèle que nous avons défini est basé sur celui de XPDL que nous avons étendu afin de supporter la gestion du temps et les activités coopératives (via la notion de tâche) ainsi que pour favoriser la réutilisation des modèles d'activité.

Sur ce moteur de Workflow générique particulièrement adapté aux contraintes de la FOAD, nous avons ensuite montré à travers l'utilisation de IMS-LD la démarche permettant l'exécution de scénarios pédagogiques. Cette dernière implique de définir une projection des concepts de l'EML vers le langage de définition de processus et d'implanter un ensemble de services associés au moteur de Workflows pour gérer les propriétés qui ne sont pas liées à l'exécution proprement dite (e.g., les objectifs pédagogiques). Outre la projection de IMS-LD, nous avons défini une architecture pour la gestion de ces propriétés liées à l'éducation et nous présentons un exemple de mise en œuvre : le Guide d'Apprentissage Interactif Numérique (GAIN).

Le moteur de Workflow proprement dit est constitué de 12 composants de session et de 21 composants entité, le tout représentant environ 42000 lignes de codes. Il fonctionne sur les conteneurs d'EJB Jonas et JBoss. Il a été récemment intégré comme projet au sein de la forge du consortium ObjectWeb ce qui permettra nous l'espérons de valider ses propriétés et ses fonctionnalités dans d'autres cadres.

7.1 Perspectives

Outre les perspectives d'amélioration à cours terme, on peut envisager des perspectives à ce travail dans le cadre initial de la formation d'une part et d'un point de vue plus fondamental sur l'approche par méta-modèle que nous avons entamé d'autre part.

7.1.1 Perspectives par rapport à la formation

L'objectif de notre travail est de permettre d'intégrer les ressources pédagogiques au sein d'activités et de scénarios d'apprentissage afin d'améliorer l'acquisition des connaissances, il reste un énorme travail sur les usages afin de valider ce postulat et de trouver les « bons scénarios ». A plus court terme, COW devrait (si le projet est retenu...) être utilisé au sein de l'Université de Lille 1 pour la gestion des parcours, c'est-à-dire gérer le processus d'inscription des étudiants ainsi que l'évolution dans les différentes formations. On se trouve ici dans un cadre différent puisqu'on ne s'intéresse plus ici aux modules mais à un plus gros grain d'enchaînement de modules. Enfin dans le cadre de la mise en place du LMD (Licence-Master-Doctorat) et des crédits ECTS (European Credit Training System) il serait intéressant

d'étendre les fonctionnalités de COW pour gérer les compétences des étudiants et les pré-requis afin de construire des parcours individualisés. Un autre objectif concerne le support de la mobilité des utilisateurs (ou M-learning). Nous avons montré que l'accès à la plate-forme pouvait être réalisé depuis n'importe quel type de terminal (PC, PDA, GSM, ...) connecté à notre plate-forme. Nous devons étendre ce support à l'exécution des processus dans un contexte déconnecté, où certaines parties du processus d'apprentissage peuvent se dérouler sans connexion permanente aux serveurs.

7.1.2 Perspectives par rapport à la méta-modélisation

Pour réaliser le passage d'un EML vers notre langage, nous avons utilisé une comparaison des méta-modèles. Ensuite la transformation est réalisée par des filtres XSL. Notre approche pourrait être simplifiée par l'exploitation d'un référentiel MOF (*Meta-Object Facility*) contenant les méta-modèles des différents langages et l'utilisation de QVT (*Query/View/Transformation*). A court terme, nous testerons ModFact [Mod03] afin de remplacer dans notre architecture les transformateurs COWL2IMSLD et IMSLD2COWL.

Cette démarche pourra être élargie à d'autre domaines d'application comme par exemple aux langages d'ordonnancement des services webs comme par exemple WSFL (Web Service Flow Language) ou aux langages de description des processus de développement logiciel comme SPEM (Software Process Engineering Metamodel) [OMG02]. COW pourrait alors servir de support au développement d'applications.

« Du danger et de l'insécurité permanente vient la force qui pousse l'Humanité à de nouvelles et toujours plus difficiles conquêtes. »

Isaac Asimov, *La fin de l'Éternité*

Annexe A

Modèle IMS-LD

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by Colin Tattersall
  (Open University of the Netherlands) -->
<imscp:manifest xmlns:imscp="http://www.imsglobal.org/xsd/imscp_v1p1"
                xmlns:imsld="http://www.imsglobal.org/xsd/imsld_v1p0"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.imsglobal.org/xsd/imscp_v1p1
                                    http://www.imsglobal.org/xsd/imscp_v1p1p3.xsd
                                    http://www.imsglobal.org/xsd/imsld_v1p0
                                    http://www.imsglobal.org/xsd/imsld_level_a_v1p0.xsd"
                identifier="CP-Boeing-Simplified">
    <imscp:organizations>
        <imsld:learning-design identifier="LD-boeing-simplified" uri="URI" level="A">
            <imsld:title>Boeing Fuel Valve Removal simplified</imsld:title>
            <imsld:learning-objectives>
                <imsld:title>Learning objectives title</imsld:title>
                <imsld:item identifier="LOB-learning-objectives"
                            identifierref="RES-learning-objectives">
                    <imsld:title>Learning objective title</imsld:title>
                </imsld:item>
            </imsld:learning-objectives>
            <imsld:prerequisites>
                <imsld:title>Prerequisites title</imsld:title>
                <imsld:item identifier="PREQ-prerequisites" identifierref="RES-prerequisites">
                    <imsld:title>Prerequisite title</imsld:title>
                </imsld:item>
            </imsld:prerequisites>
            <imsld:components>
                <imsld:roles>
                    <imsld:learner identifier="R-learner"/>
                </imsld:roles>
                <imsld:activities>
                    <imsld:learning-activity identifier="LA-fuel-valve-lesson-intro">
                        <imsld:activity-description>
                            <imsld:title>Activity description title</imsld:title>
                            <imsld:item identifier="I-fuel-valve-lesson-intro"
                                        identifierref="RES-fuel-valve-lesson-intro">
                                <imsld:title>Intro fuel valve</imsld:title>
                            </imsld:item>
                        </imsld:activity-description>
                    </imsld:learning-activity>
                    <imsld:learning-activity identifier="LA-fuel-valve-theory">
                        <imsld:activity-description>
```

```
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-fuel-valve-theory"
                    identifierref="RES-fuel-valve-theory">
            <imsld:title>Fuel valve theory</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-lesson-hazards">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-lesson-hazards"
                    identifierref="RES-lesson-hazards">
            <imsld:title>Hazards</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-lesson-components">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-lesson-components"
                    identifierref="RES-lesson-components">
            <imsld:title>Components</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-preparation">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-preparation"
                    identifierref="RES-preparation">
            <imsld:title>Preparation</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-remove-door">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-remove-door"
                    identifierref="RES-remove-door">
            <imsld:title>Remove door</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-remove-transmitter">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-remove-transmitter"
                    identifierref="RES-remove-transmitter">
            <imsld:title>Remove transmitter</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-remove-valve">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-remove-valve"
                    identifierref="RES-remove-valve">
            <imsld:title>Remove valve</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
```

```
<imsld:learning-activity identifier="LA-knowledge-test-hazards">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-knowledge-test-hazards"
                    identifierref="RES-knowledge-test-hazards">
            <imsld:title>Test hazards</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-knowledge-test-components">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-knowledge-test-components"
                    identifierref="RES-knowledge-test-components">
            <imsld:title>Test components</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:learning-activity identifier="LA-performance-test">
    <imsld:activity-description>
        <imsld:title>Activity description title</imsld:title>
        <imsld:item identifier="I-performance-test"
                    identifierref="RES-performance-test">
            <imsld:title>Performance test</imsld:title>
        </imsld:item>
    </imsld:activity-description>
</imsld:learning-activity>
<imsld:activity-structure identifier="AS-introduction"
                          number-to-select="2"
              structure-type="sequence">
    <imsld:title/>
    <imsld:learning-activity-ref ref="LA-fuel-valve-lesson-intro"/>
    <imsld:learning-activity-ref ref="LA-fuel-valve-theory"/>
</imsld:activity-structure>
<imsld:activity-structure identifier="AS-fuel-valve-lessons"
                          number-to-select="2"
              structure-type="selection">
    <imsld:title/>
    <imsld:learning-activity-ref ref="LA-lesson-hazards"/>
    <imsld:learning-activity-ref ref="LA-lesson-components"/>
</imsld:activity-structure>
<imsld:activity-structure identifier="AS-fuel-valve-removal-procedure"
                          number-to-select="4"
              structure-type="sequence">
    <imsld:title/>
    <imsld:learning-activity-ref ref="LA-preparation"/>
    <imsld:learning-activity-ref ref="LA-remove-door"/>
    <imsld:learning-activity-ref ref="LA-remove-transmitter"/>
    <imsld:learning-activity-ref ref="LA-remove-valve"/>
</imsld:activity-structure>
<imsld:activity-structure identifier="AS-lessons-and-procedure"
                          number-to-select="2"
              structure-type="sequence">
    <imsld:title/>
    <imsld:environment-ref ref="E-interactive-electronic-training-manual"/>
    <imsld:activity-structure-ref ref="AS-fuel-valve-lessons"/>
    <imsld:activity-structure-ref ref="AS-fuel-valve-removal-procedure"/>
</imsld:activity-structure>
<imsld:activity-structure identifier="AS-tests"
                          number-to-select="2"
              structure-type="sequence">
```

```
                    <imsld:title/>
                    <imsld:environment-ref ref="E-interactive-electronic-training-manual"/>
                    <imsld:learning-activity-ref ref="LA-knowledge-test-hazards"/>
                    <imsld:learning-activity-ref ref="LA-knowledge-test-components"/>
                    <imsld:learning-activity-ref ref="LA-performance-test"/>
                </imsld:activity-structure>
                <imsld:activity-structure identifier="AS-boeing-simplified"
                                          number-to-select="3"
                                structure-type="sequence">
                    <imsld:title/>
                    <imsld:activity-structure-ref ref="AS-introduction"/>
                    <imsld:activity-structure-ref ref="AS-lessons-and-procedure"/>
                    <imsld:activity-structure-ref ref="AS-tests"/>
                </imsld:activity-structure>
            </imsld:activities>
            <imsld:environments>
                <imsld:environment identifier="E-interactive-electronic-training-manual">
                    <imsld:title>Interactive Electronic Technical Manual</imsld:title>
                </imsld:environment>
            </imsld:environments>
        </imsld:components>
        <imsld:method>
            <imsld:play identifier="PLAY-Boeing-simplified" isvisible="true">
                <imsld:act identifier="ACT-individualized-learning">
                    <imsld:role-part identifier="RP-individualized-learning">
                        <imsld:role-ref ref="R-learner"/>
                        <imsld:activity-structure-ref ref="AS-boeing-simplified"/>
                    </imsld:role-part>
                    <imsld:complete-act>
                        <imsld:when-role-part-completed ref="RP-individualized-learning"/>
                    </imsld:complete-act>
                </imsld:act>
                <imsld:complete-play>
                    <imsld:when-last-act-completed/>
                </imsld:complete-play>
            </imsld:play>
        </imsld:method>
    </imsld:learning-design>
</imscp:organizations>
<imscp:resources>
    <imscp:resource identifier="RES-knowledge-test-hazards" type=""/>
    <imscp:resource identifier="RES-knowledge-test-components" type=""/>
    <imscp:resource identifier="RES-performance-test" type="webcontent"/>
    <imscp:resource identifier="RES-CourseFailed" type="webcontent"/>
    <imscp:resource identifier="RES-learning-objectives" type=""/>
    <imscp:resource identifier="RES-prerequisites" type=""/>
    <imscp:resource identifier="RES-fuel-valve-lesson-intro" type="webcontent"/>
    <imscp:resource identifier="RES-fuel-valve-theory" type="webcontent"/>
    <imscp:resource identifier="RES-lesson-hazards" type="webcontent"/>
    <imscp:resource identifier="RES-lesson-components" type="webcontent"/>
    <imscp:resource identifier="RES-preparation" type="webcontent"/>
    <imscp:resource identifier="RES-remove-door" type="webcontent"/>
    <imscp:resource identifier="RES-remove-transmitter" type="webcontent"/>
    <imscp:resource identifier="RES-remove-valve" type="webcontent"/>
</imscp:resources>
</imscp:manifest>
```

Glossaire

API : Application Programming Interface

CBT : Computer Based Training
CMI : Computer Managed Instruction
CSCL : Computer Supported Cooperative Learning

DARE : Distributed Activities in a Reflexive Environment. DARE [Bou00] est un méta-groupware réflexif développé au sein du laboratoire TRIGONE.
DTMF : Dual Tone Multi Frequency

EIAH : Environnement Informatique pour l'Apprentissage Humain
EML : Educational Modelling Languages

FOAD : Formation Ouverte et Á Distance

GED : Gestion Électronique de Documents
GSM : Global System for Mobile communications

HTML : Hyper Text Markup Language

IDL : Interface Description Language

J2EE : Java 2 Enterprise Edition
JMS : Java Message Service
JMX : Java Management Extensions

LMML : Learning Material Markup Language
LMS : Learning Management System

OMG : Object Management Group

PDA : Personal Digital Assistant
PTM : Passau Teachware Model

RFP : Request For Proposal

SOAP : Simple Object Access Protocol
SSII : Société de Services et d'Ingénierie en Informatique

TCAO : Travail Coopératif Assisté par Ordinateur
TML : Tutorial Markup Language

UDDI : Universal Description, Discovery and Integration
UML : Unified Modelling Language

WAP : Wireless Application Protocol
WfMC : Workflow Management Coalition
WML : Wireless Markup Language
WSDL : Web Service Description Language
WSFL : Web Service Flow Language

Index

Bibliographie

[AIC95] AICC. Courseware Interchange. AICC Guidelines & Recommendations AGR-007 version 1.0, Aviation Industry CBT Commitee, 29 août 1995.

[AIC98] AICC. Computer Managed Instruction. AICC Guidelines & Recommendations AGR-006 version 2.0, Aviation Industry CBT Commitee, 19 mai 1998.

[AIC01] AICC. CMI Guidelines for Interoperability. Technical Report CMI001, Revision 3.5, Aviation Industry CBT Commitee, 2 avril 2001.

[AIC02] AICC. Courseware Delivery Stations. AICC Guidelines & Recommendations AGR-002 version 9.1, Aviation Industry CBT Commitee, 06 février 2002.

[AIS+77] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language*. Oxford University Press, New York, USA, 1977.

[Arc03a] Archimed SA. Site web, 2003. `http://www.archimed.fr`.

[Arc03b] Model Driven Architecture. Site Web, 2003. `http://www.omg.org/mda`.

[Avi03] Aviation Industry CBT Commitee. Site web, 2003. `http://www.aicc.org`.

[BBC+01] S. Breitenbach, T. Burd, N. Chidambaram, E. Astrid Anderson, X. Tang, P. Houle, D. Newsome, and X. Zhu. *Early Adopter VoiceXML*. Wrox, 2001.

[BD00] Grégory Bourguin and Alain Derycke. A Reflective CSCL Environment with Foundations Based on the Activity Theory. In *Intelligent Tutoring Systems, 5th International Conference*, volume 1839 of *Lecture Notes in Computer Science*, pages 272–281, Montréal, Canada, juin 2000. Springer.

[BDT01] Grégory Bourguin, Alain Derycke, and Jean-Claude Tarby. Beyond the Interface : Co-evolution Inside Interactive Systems - A Proposal Founded on Activity Theory. In Jean Vanderdonckt Ann Blandford and Phil Gray, editors, *People and Computers XV - Interaction without Frontiers*, pages 297 – 310, Lille, France, septembre 2001. Springer. ISBN 1-85233-515-7.

[BG01] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *16th IEEE International Conference on Automated Software Engineering (ASE 2001)*, pages 273–280, Coronado Island, San Diego, CA, USA, novembre 2001. IEEE Computer Society. ISBN : 0-7695-1426-X.

[BGW93] Daniel G. Bobrow, R.G. Gabriel, and J.L. White. *CLOS in Context - The Shape of the Design Space*, chapter 2, In Object Oriented Programming - The CLOS Perspective. MIT Press, 1993.

[Blo56] Benjamin Bloom. *Taxonomy of educational objectives : The classification of educational goals : Handbook I, cognitive domain.* Longman, New-York, 1956.

[BM97] Gregory Bedny and David Meister. *The Russian Theory of Activity : Current Applications to Design and Learning.* Lawrence Erlbaum Associates, 1997. ISBN : 0-8058-1771-9.

[Bou00] Gégory Bourguin. *Un support informatique à l'activité coopérative fondé sur la Théorie de l'activité : le projet DARE.* Thèse de Doctorat en Informatique, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, 13 juillet 2000.

[Bri] Dan Brickley. Towards an open question-interchange framework. Technical report, University of Bristol. `http://www.ilrt.bris.ac.uk/netquest/about/lang/motivation.html`.

[Cas98] Fabio Casati. A discussion on approaches to handling exceptions in workflows. In *Towards Adaptive Workflow Systems, CSCW-98 Workshop*, 1998. `http://ccs.mit.edu/klein/cscw98/paper26/`.

[Che02] Vincent Chevrin. Multimodalité et Personnalisation dans les systèmes Interactifs Evolutifs : Adaptation dynamique au contexte de la tâche des utilisateurs. Mémoire de DEA Informatique, Laboratoire TRIGONE, équipe NOCE, Villeneuve d'Ascq, France, Juillet 2002.

[CP99] Fabio Casati and Giuseppe Pozzi. Modeling exceptional behaviors in commercial workflow management systems. In *International conference on cooperative information systems COOPIS'99*, pages 127–138, 2-4 septembre 1999.

[DGL00] Wolfgang Deiters, Thomas Goesmann, and Thorsten Löffeler. Flexibility in workflow management dimensions and solutions. *International journal of computer systems science & engineering*, 15(5) :303–314, septembre 2000. Special issue : Flexible workflow technology driving the networked economy.

[DmYK01] Linda G. DeMichiel, L. Ümit Yalçinalp, and Sanjeev Krishnan. Enterprise JavaBeans^{TM} Specification, Version 2.0. Technical Report Final Release, Sun Microsystems, Inc., aout 2001.

[DST03] DSTC. Distributed systems technology centre (dstc) homepage, 2003. `http://www.dstc.edu.au`.

[EGL98] Johann Eder, Herbert Groiss, and Walter Liebhart. The workflow management system panta rhei. In A. Dogac, L. Kalinichenko, T. öszu, and A. Sheth, editors, *Workflow Management Systems and Interoperability.* Springer-Verlag, 1998. `http://www.ifi.uni-klu.ac.at/Publications/pubfiles/psfiles/1998-0003_EdGL.ps`.

[EL95] Johann Eder and Walter Liebhart. The Workflow Activity Model WAMO. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIs'95)*, Wien, Autriche, mai 1995.

[Eng87] Engeström. *Learning By expanding : an activity-theorical approach to developmental research.* Orienta-konsultit Oy, Helsinki, Finlande, 1987.

[Enh03] Enhydra JaWE. Site web, 2003. http://jawe.enhydra.org/.

[EP00] Johann Eder and Euthimios Panagos. Managing Time in Workflow Systems. In Layne Fischer (ed.), editor, *Workflow Handbook 2001, Future Strategies INC. in association with Workflow Management Coalition*, pages 109–132, 2000. ISBN 0-9703509-0-2, http://www.ifi.uni-klu.ac.at/Publications/pubfiles/pdffiles/2000-0101-JEEP.pdf.

[EPPR99] Johann Eder, Euthimios Panagos, Heinz Pozewaunig, and Michael Rabinovich. Time Management in Workflow Systems. In M.E. Orlowska W.Abramowicz, editor, *BIS'99 3rd International Conference on Business InformationSystems*, pages 265–280, Poznan, Pologne, 1999. Springverlag. ISBN : 1-85233-167-4, http://www.ifi.uni-klu.ac.at/Publications/pubfiles/psfiles/1999-0004-EPPR.ps.

[EtH98] David Edmond and Arthur H.M. ter Hofstede. Achieving Workflow Adaptability by means of Relfection. In *Towards Adaptive Workflow Systems, CSCW-98 Workshop*, 1998. http://ccs.mit.edu/klein/cscw98/paper29/paper.ps.

[Eva01] Tom Evans. New research challenges for technology supported learning. Technical report, Information Society Technologies (IST), Commission Européenne, 2001.

[GAC+97] Esin Gokkoca, Mehmet Altinel, Ibrahim Cingil, E.Nesime Tatbul, Pinar Koksal, and Asuman Dogac. Design and implementation of a distributed workflow enactment service. In *Proceedings of International Conference on Cooperative Information Systems (COOPIS'97)*, june 1997. ftp://ftp.srdc.metu.edu.tr/pub/metuflow/papers/coopis97.ps.gz.

[GG98] T. C. Nicholas Graham and John C. Grundy. External requirements of groupware development tools. In Prasun Dewan Stephane Chatty, editor, *Engineering for Human-Computer Interaction, Seventh Working Conference on Engineering for Human-Computer Interaction*, pages 363–376, Heraklion, Crete, Greece, 14-18 septembre 1998. ISBN 0-412-83520-7.

[GGM98] Jean-Marc Geib, Christophe Gransart, and Philippe Merle. *Corba : Des concepts à la pratique*. InterEdition, décembre 1998. http://corbaweb.lifl.fr.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN : 0-201-63361-2.

[Goi03] Grégory Goidin. GAIN et Divilab/Task Manager. Stage DESS, Universisté des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, septembre 2003.

[HA98] Claus Hagen and Gustavo Alonso. Flexible Exception Handling in the OPERA Process Support System. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, Amsterdam, Pays-Bas, mai 1998.

[HA00] Claus Hagen and Gustavo Alonso. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering*, 26(10) :943–958, octobre 2000.

[Hoo95] Frédéric Hoogstoel. *Une approche organisationnelle du travail coopératif assisté par ordinateur : application au projet co-learn*. Thèse de Doctorat en Informatique, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, 1995.

[HSB98] Yanbo Han, Amit Sheth, and Christoph Bussler. A Taxonomy of Adaptive Workflow Management. In *Workshop of the 1998 ACM Conference on Computer Supported Cooperative Work*, Seatle, Washington, USA, novembre 1998. `http://ccs.mit.edu/klein/cscw98/paper03`.

[IMS02a] IMS. IMS Question & Test Interoperability : An Overview. Final Specification Version 1.2, IMS Global Learning Consortium, 11 février 2002. `http://www.imsproject.org/question/index.cfm`, `http://www.imsproject.org/question/qtiv1p2/imsqti_oviewv1p2.html`.

[IMS02b] IMS. IMS Question & Test Interoperability : ASI Information Model Specification. Final Specification Version 1.2, IMS Global Learning Consortium, 11 février 2002.

[IMS02c] IMS. IMS Question & Test Interoperability : ASI Selection & Ordering. Final Specification Version 1.2, IMS Global Learning Consortium, 11 février 2002.

[IMS02d] IMS. IMS Question & Test Interoperability : ASI XML Binding Specification. Final Specification Version 1.2, IMS Global Learning Consortium, 11 février 2002. `http://www.imsproject.org/question/qtiv1p2/imsqti_asi_bindv1p2.html`.

[IMS03a] IMS. Ims content packaging specification home page, 2003. `http://www.imsproject.org/content/packaging/index.cfm`.

[IMS03b] IMS. IMS Learning Design Information Model. Final Specification, IMS Global Learning Consortium, 20 january 2003. `http://www.imsproject.org/learningdesign/ldv1p0/imsld_infov1p0.html`.

[IMS03c] IMS. Ims learning design specification home page, 2003. `http://www.imsproject.org/learningdesign/index.cfm`.

[JBR00] Ivar Jacobson, Grady Booch, and James Rumbaugh. *Le processus unifié de développement logiciel*. Eyrolles, juin 2000. ISBN : 2-212-09142-7.

[JMX03] JMX. Site web, 2003. `http://java.sun.com/jmx`.

[JW97] Ralph E. Johnson and Booby Woolf. *The Type Object Pattern*, chapter 4, in Pattern Languages of Program Design 3. Software Patterns Series. Addison Wesley, octobre 1997.

[KAD98] Pinar Koksal, Sena Nural Arpinar, and Asuman Dogac. Workflow history management. *Sigmod Record*, 27(1), march 1998. `ftp://dtp.srdc.metu.edu.tr/pub/metuflow/papers/sonsig.ps.gz`.

[KCD99] Pinar Koksal, Ibrahim Cingil, and Asuman Dogac. A component-based workflow system with dynamic modifications. In Springer-Verlag, editor, *Proceedings of Next Generation Information Technologies and Systems (NGITS'99)*, volume 1649 of *Lecture Notes in Computer Science*, pages 238–255, Israel, July 1999. `http://www.srdc.metu.edu.tr/papers/ngits99.ps`.

[KdRB91] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. The MIT Press, 1991. ISBN : 0-262-61074-4.

[KG99] Markus Kradolfer and Andreas Geppert. Dynamic Workflow Schela Evolution Based on Workflow type Versioning ans Workflow Migration. In *International Conference on Cooperative Information Systems COOPIS'99*, pages 104–114, 2-4 septembre 1999.

[Kic96] Gregor Kiczales. Beyond the black box : open implementation. *IEEE Software*, janvier 1996. `http://www2.parc.com/csl/groups/sda/projects/oi/ieee-software/`.

[KLL+97] Gregor Kiczales, John Lamping, Cristina Videira Lopes, Chris Maeda, Anurag Mendhekar, and Gail Murphy. Open Implementation Design Guidelines. In *International Conference on Software Engineering*, pages 481 – 490, 1997.

[Kop00] Rob Koper. From change to renewal : Educational technology foundations of electronic environments, 15 décembre 2000. `http://eml.ou.nl/introduction/docs/koper-inaugural-address.pdf`.

[Kop01] Rob Koper. Modeling units of study from a pedagogical perspective : the pedagogical meta-model behind EML. Technical report, Educational TEchnology Expertise Centre, Open University of the Netherlands, Juin 2001. `http://eml.ou.nl/introduction/docs/ped-metamodel.pdf`.

[Kra00] Markus Kradolfer. *A Workflow Metamodel Supporting Dynamic Reuse-Based Model Evolution*. PhD thesis, Der Wirtschaftwissenschaftlichen Fakultät der Universität Zürich, mai 2000.

[LASS00] A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler. The wise approach to electronic commerce. *International Journal of Computer Systems Science & Engineering, special issue on Flexible Workflow Technology Driving the Networked Economy*, 15(5), september 2000. `http://www.inf.ethz.ch/department/IS/iks/publications/lass00.html`.

[LHSO02] Joe Lin, Charley Ho, Wasim Sadiq, and Maria E. Orlowska. Using Workflow Technology to Manage Flexible e-learning Services. *Journal of Educational Technology & Society*, 5(4), octobre 2002. IEEE Learning Technology Task Force.

[LMM03] LMML. Learning material markup language, 2003. `http://www.lmml.de`.

[LP01] Miltiadis D. Lytras and Athanasia Pouloudi. E-learning : Just a Waste of Time. In *Proceedings of the Seventh Americas Conference on Information Systems (AMCIS 2001)*, pages 216–222, Boston, Massachusetts, USA, 3-5 Août 2001.

[LS97] Yu Lei and Munindar P. Singh. A comparison of workflow metamodels. In *Proceedings of the ER-97 Workshop on Behavioral Modeling and Design Transformations : Issues and Opportunities in Conceptual Modeling*, Los Angeles, novembre 1997. `http://osm7.cs.byu.edu/ER97/workshop4/ls.html`.

[LSAS01] A. Lazcano, H. Schuldt, G. Alonso, and H.-J. Schek. Wise : Process based e-commerce. *IEEE Data Engineering Bulletin, Special Issue on Infrastructure for*

Advanced E-services, 24 (1), march 2001. `http://www.inf.ethz.ch/department/IS/iks/publications/lsas01.html`.

[Mae87] Pattie Maes. *Computational Reflection*. PhD thesis, Artificial Intelligence Laboratory, Vrije Universiteit, Brussel, Belgium, 1987.

[Man00] Dragos A. Manolescu. *Micro-Workflow : A Workflow Architecture Supporting Compositional Object-Oriented Software development*. PhD thesis, University of Illinois, October 2000. `http://micro-workflow.com/PhDThesis`.

[Man03] Dragos A. Manolescu. *An Extensible Workflow Architecture with Objects and Pattern*, chapter 4, in Technology of Object-Oriented Languages, Systems, and Architecture. Kluwer Academic Publishers, 2003. `http://micro-workflow.com/PDF/toolsee01.pdf`.

[Met03] Metamodel.com. Site web, 2003. `http://www.metamodel.com`.

[MJ98] Dragos A. Manolescu and Ralph E. Johnson. Patterns of Workflow Management Facility. `http://micro-workflow.com/PDF/PWFMF.pdf`, november 1998.

[MJ00] Dragos A. Manolescu and Ralph E. Johnson. A micro-workflow component for federated workflow. `http://micro-workflow.com/PDF/oopsla2k.pdf`, octobre 2000.

[MLMK97] Chris Maeda, Arthur Lee, Gail Murphy, and Gregor Kiczales. Open Implementation Analysis and Design. In *Symposium on Software Reuse*, 1997. `http://www2.parc.com/csl/groups/sda/projects/oi-at-parc/ourpapers/oiad.pdf`.

[MO00] Olivera Marjanovic and Maria E. Orlowska. Making Flexible Learning More Flexible. In *IEEE International Workshop on Advanced Learning Technologies IWALT'2000*, New Zealand, December 2000. `http://flex-el.com`.

[Mod03] ModFact. Site web, 2003. `http://modfact.lip6.fr/ModFactWeb/index.jsp`.

[MS01] Vlada Matena and Beth Stearns. *Applying Enterprise JavaBeans : Component-Based Development for the J2EE Platform*. Addison-Wesley, march 2001. ISBN 0201702673.

[Mul99] Pierre-Alain Muller. *Modélisation objet avec UML*. Eyrolles, 1999. ISBN : 2-212-08966-X.

[Nar95] Bonnie A. Nardi, editor. *Context and Consciousness : Activity Theory and Human-Computer Interaction*. MIT Press, novembre 1995. ISBN : 0262140586.

[Obj03] ObjectWeb. Site web, 2003. `http://www.objectweb.org`.

[OMG98] OMG. Common Object Request Broker : Architecture and Specification. OMG TC Document formal/98-07-01, Object Management Group, février 1998.

[OMG00a] OMG. *Persistent Object Service (POS) Specification*. Object Management Group, 2000.

[OMG00b] OMG. Workflow Management Facility Specification, version 1.2. OMG TC Document formal/00-05-02, Object Management Group, avril 2000. `http://www.omg.org/docs/formal/00-05-02.pdf`.

[OMG02] OMG. Software Process Engineering Metamodel Specification, version 1.0. OMG TC Document formal/02-11-14, Object Management Group, novembre 2002. http://www.omg.org/docs/formal/02-11-14.pdf.

[Rag03] Alain Ragot. GAIN : Guide d'Apprentissage Interactif Numérique. Projet DESS, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, avril 2003.

[RD97] Manfred Reichert and Peter Dadam. A framework for Dynamic Changes in Workflow Management Systems. In *Proceedings of DEXA'97*, Toulouse, France, 1-2 septembre 1997.

[Riv02] Michel Riveill. SOAP. *Techniques de l'Ingénieur, traité Informatique, H2740*, mai 2002. http://www.techniques-ingenieur.fr/affichage/DispIntro.asp?nGcmId=h2740.

[RvRK+02] Adrian Rawlings, Peter van Rosmalen, Rob Koper, Miguel Rodriguez-Artacho, and Paul Lefrere. Survey of Educational Modelling Languages (EMLs). Learning Technologies Workshop Version 1, CEN/ISSS WS/LT, 19 septembre 2002.

[Sad00] Shazia Sadiq. Handling Dynamic Schema Change in Process Models. In *Proceedings of the 11th Australian Database Conference*, Canberra, Australie, 30 janvier - 3 février 2000. http://www.dstc.edu.au/praxis/publications/ssadiq_adc_2000.pdf.

[Sea69] Searle. *Speech Acts : An essay in the Philosophy of Language*. Cambridge University Press, 1969.

[SF02] Christian Süß and Burkhard Freitag. LMML – The Learning Material Markup Language Framework. In *International Workshop ICL*, Villach, Autriche, Septembre 2002.

[SFB99] Christian Süß, Burkhard Freitag, and Peter Brössler. Metamodeling for Web-Based Teachware Managment. In P.P. Chen, D.W. Embley, J. Kouloumdijan, S.W. Liddle, and J.F. Roddick, editors, *Advances in Conceptual Modeling. ER'99 Workshop on the World-Wide Web and Conceptual Modeling, Paris, France, Nov. 1999*, volume 1727 of *LNCS*, pages 360–373, Berlin, 1999. Springer-Verlag.

[SGJ+96] Amit Sheth, Dimitrios Georgakopoulos, Stef M.M. Joosten, Marek Rusinkiewicz, Walt Scacchi, JAck Wileden, and Alexander Wolf. Report from the NSF Workshop on Workflow and Process Automation in Information Systems. Technical Report UGA-CS-TR-96-003, Dept. of Computer Sc., University of Georgia, octobre 1996.

[SME99] Shazia W. Sadiq, Olivera Marjanovic, and Maria E.Orlowska. Managing Change and Time in Dynamic Workflow Processes. *International Journal of Cooperative Information Systems*, 1999. http://citeseer.nj.nec.com/olivera99managing.html.

[SW03] Birgit Schmidt-Wesche. *IBM WebSphere Platform User Roles*. IBM, 2003.

[Tan94] Andrew Tanenbaum. *Systèmes d'exploitation : Systèmes centralisés, systèmes distribués*. Informatique Intelligence Artificielle. InterEdition, Paris, 1994. ISBN : 2-7296-0706-4.

[Tar03] Targeteam. Site web, 2003. `http://www.targeteam.org`.

[Tee02a] Gunnar Teege. *Documentation of Targeteam V O.5.6.* Institut für Informatik Technische Universität München et Institut für Informationstechnische Systeme Universität des Bundeswehr München, juillet 2002.

[Tee02b] Gunnar Teege. Reuse of Teaching Materials in Targeteam. In *International Workshop on Interactive Computer aided Learning ICL 2002*, Villach, Autriche, 2002.

[TMLa] Bristol University. *Tutorial Markup Language DTD.* `http://www.ilrt.bris.ac.uk/netquest/liveserver/TML_INSTALL/lib/ETS/dtd/TML_4.0`.

[TMLb] Bristol University. *Tutorial Markup Language (TML).* `http://www.ilrt.bris.ac.uk/netquest/liveserver/TML_INSTALL/doc/tml_user.html`.

[Tom99] Dimitrios Tombros. *An Event- and Repository-Based Component Framework for Workflow System Architecture.* PhD thesis, Der Wirtschaftwissenschaftlichen Fakultät der Universität Zürich, novembre 1999.

[UDD] UDDI. Universal Description, Discovery and Integration of Web Services. `http://www.uddi.org`.

[VD97] Claude Viéville and Alain Derycke. Self organised group activities supported by asynchronous structured conversations. In Felisa Verdejo and Gordon Davies, editors, *Proceedings of the 1997 IFIP TC3/WG3.3&3.6, Join working conference, The Virtual Campus : Trends for Higher Education and Trainig*, pages 175 – 188, Madrid, Espagne, 27-29 novembre 1997. Chapman & Hall.

[vdA99] W. M. P. van der Aalst. Generic Workflow Models : How to handle Dynamic Change and Capture Management Information ? In *International Conference on Cooperative Information Systems COOPIS'99*, pages 115–126, 2-4 septembre 1999.

[Vié98] Claude Viéville. An Asynchronous collaborative learning system on the web. In Stephen Hailes Reza Hazemi and S Wilbur, editors, *The Digital University - reinventing the Academy*, pages 99 – 113. Springer, 1998.

[Vié02] Claude Viéville. Learning Activities in a Virtual Campus. In Reza Hazemi and Stephen Hailes, editors, *The Digital University - Building a Learning Community*, CSCW, chapter 15, pages 215 – 227. Springer, 2002. ISBN : 1-85233-478-9.

[Voi03] VoiceXML Forum, 2003. `http://www.voicexml.org`.

[VP01] Thomas Vantroys and Yvan Peter. A WMF-based Workflow For E-learning. In *European Research Seminar on Advances in Distributed Systems (ERSADS 2001)*, Bertinoro (Forli), Italia, 14 - 18 May 2001. `http://noce.univ-lille1.fr/~tvantroys/publication/ERSADS2001_vantroys_peter.pdf`.

[VP02] Claude Viéville and Yvan Peter. Learning Activity Modelling and Management. In *International Conference on Computer in Education (ICCE'2002)*, volume 1, pages 288 – 292, Auckland, New Zealand, décembre 2002. IEEE Computer Society. ISBN : 0-7695-1509-6.

[VR02] Thomas Vantroys and José Rouillard. Workflow and Mobile Devices in Open Distance Learning. In *IEEE International Conference on Advanced Learning Technologies ICALT 2002*, pages 123 – 126, Kazan, Tatarstan, Russie, 9 - 12 septembre 2002. `http://lttf.ieee.org/icalt2002/`.

[W3C00] W3C. Simple Object Access Protocol (SOAP). `http://www.w3.org/TR/SOAP/`, 8 mai 2000.

[W3C01] W3C. Web Services Description Language 1.1. `http://www.w3.org/TR/wsdl`, 15 mars 2001.

[WAP03] Wireless Application Protocol, 2003. `http://www.wapforum.org`.

[WF86] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1986.

[WfM03] 2003. Workflow Management Coalition, `http://www.wfmc.org`.

[WID03] WIDE. site web, 2003. `http://dis.sema.es/projects/WIDE`.

[Wor95] Workflow Management Coalition. The Workflow Reference Model. Technical Report WfMC-TC-1003, Version 1.1, Workflow Management Coalition, 19 janvier 1995. `http://www.wfmc.org/standards/docs/tc003v11.pdf`.

[Wor98a] Workflow Management Coalition. Audit Data Specification. Technical Report WfMC-TC-1015, Version 1.1, Workflow Management Coalition, 22 septembre 1998.

[Wor98b] Workflow Management Coalition. Workflow Management Application Programming Interface (Interface 2&3) Specification. Technical Report WfMC-TC-1009, Version 2.0, Workflow Management Coalition, Juillet 1998.

[Wor99a] Workflow Management Coalition. Interface 1 : Process Definition Interchange - Process Model. Technical Report WfMC-TC-1016-P, Version 1.1, Workflow Management Coalition, 29 octobre 1999.

[Wor99b] Workflow Management Coalition. Terminology & Glossary. Technical Report WfMC-TC-1011, Version 3.0, Workflow Management Coalition, février 1999.

[Wor01] Workflow Management Coalition. Workflow Standard - Interoperability Wf-XML Binding. Specification WFMC-TC-1023, version 1.1, Workflow Management Coalition, 14 octobre 2001. `http://www.wfmc.org/standards/docs/Wf-XML-11.pdf`.

[Wor02] Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Language. Specification WFMC-TC-1025, version 1.0, Workflow Management Coalition, 25 octobre 2002. `http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf`.

[WSKF02] Franz Weitl, Christian Süß, Rudolf Kammerl, and Burkhard Freitag. Presenting Complex e-Learning Content on the Web : A Didactical Reference Model. In *Proceedings of E-Learn 2002*, 2002.

[Zis77] Zismann. *Representation, Specification and Automation of Office Procedures*. PhD thesis, Wharton School of Business, 1977.

Résumé

La standardisation des ressources pédagogiques étant bien avancée, l'effort se porte maintenant vers la définition de langages (Educational Modelling Languages) permettant l'expression et la réutilisation de scénarios pédagogiques reposant sur ces ressources. En effet, les plates-formes ne se contentent plus de permettre l'accès aux ressources mais cherchent à supporter la gestion de véritables parcours de formation complets. La technologie des systèmes de workflows est particulièrement adaptée à la coordination des activités. Cependant, dans le cadre d'activités humaines par définition imprédictible, ces systèmes doivent être très flexibles afin de s'adapter aux différents utilisateurs et au contexte Notre objectif est de proposer une plate-forme d'exécution de scénarios pédagogiques en se basant sur une approche par système de workflows. Cette réalisation passe dans un premier temps par l'étude des langages de scénarios pédagogiques et des systèmes de workflows afin notamment de réaliser des rapprochements permettant le passage d'une modélisation à l'autre. Ce point est important car les acteurs qui interviendront dans le système auront des points de vues et des approches différentes. Ensuite, nous prendrons en compte les aspects concernant la malléabilité pour réaliser notre plate-forme d'exécution qui devra permettre un maximum de flexibilité pour autoriser une adaptation continue au contexte. Notre objectif n'est pas de reconstruire une nouvelle plate-forme mais bien de créer un "composant technique" réalisant l'exécution de scénarios et intégrable dans les Learning Management Systems (LMS) existants. Nous avons basé notre approche sur la programmation par composants logiciels. Elle nous permet la construction d'applications par liens entre "briques" existantes. Elle nous libère également de la gestion des aspects "non fonctionnels" (gestion de la sécurité, des transactions, ...) pour nous consacrer au développement "métier". Elle correspond à notre vision d'un LMS constitué de briques indépendantes spécialisées dans des domaines particuliers.

Mots-clés: Système de workflows, FOAD, EML, composants logiciels, méta-modèle