Environnement de procédé extensible pour l'orchestration - Application aux services web

Sonia Jamal (epouse Sanlaville)

UNIVERSITE JOSEPH FOURIER – GRENOBLE I

THESE

pour obtenir le grade de

DOCTEUR de l'Université Joseph Fourier de Grenoble

(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Discipline : Informatique

présentée et soutenue publiquement par

Sonia JAMAL (épouse SANLAVILLE)

le 13 décembre 2005

Environnement de procédé extensible pour l'orchestration Application aux services web

Directeur de thèse :

Jacky ESTUBLIER

JURY :

Yves Chiaramella, Université Joseph Fourier, Grenoble	Président de jury
Claude Godart, Université Henri Poincaré, Nancy	Rapporteur
Michel Lemoine, CERT ONERA, Toulouse	Rapporteur
Marie-Christine Fauvet, Université Joseph Fourier, Grenoble	Examinatrice
Christer Fernstrom, XRCE - Xerox Research Centre Europe, Grenoble	Examinateur
Jacky Estublier, Directeur de recherche au CNRS, Grenoble	Directeur de thèse

Thèse préparée au sein du Laboratoire LSR – Equipe ADELE

Aux plus chers à mon cœur :
mes parents,
Monia, Dany,
Rémy

Remerciements

Je tiens à remercier,

M. Yves Chiaramella, Professeur à l'Université Joseph Fourier à Grenoble, de me faire l'honneur de présider mon jury de thèse.

M. Claude Godart, Professeur à l'Université HENRI Poincaré à Nancy et M. Michel Lemoine, Directeur de Recherche au CERT ONERA à Toulouse, pour avoir accepté de lire et d'évaluer mon travail de thèse et pour leurs retours constructifs.

Mme Marie-Christine Fauvet, Professeur à l'Université Joseph Fourier-Grenoble, pour sa participation à ce jury et pour ses conseils dans le domaine de la recherche et de l'enseignement.

M. Christer Fernstrom, directeur de recherche à Xerox Research Centre Europe – XRCE - Centre Planner à Grenoble, pour avoir accepté de participer à mon jury de thèse.

M. Jacky Estublier, Directeur de Recherche au CNRS, pour avoir dirigé ma thèse et pour ses conseils et son aide. Je lui suis très reconnaissante du point de vue professionnel.

Je tiens également à remercier

Mes amis de l'équipe Adèle. Je souhaite citer plus particulièrement Tuyet avec qui j'ai partagé quatre années de grande amitié. Merci à German et à Jorge pour leur aide, leurs conseils et leur amitié, merci à Vincent, Mikael, Anca, Cristina, Sergio, Didier, Jean-Marie, et tous les amis de l'équipe Adèle pour leur amitié et leur soutien.

Mes amis Syriens à Grenoble. Plus particulièrement Lina, avec qui j'ai partagé les difficultés de mes premières années en France, ainsi que Dima, Lamis et Samer...

Mes amis en Syrie. Mes copines de lycée Dima, Dima et Silva, et mes amis d'université Marlène, William et Luda qui m'ont soutenu malgré la distance.

Ma belle famille, pour les nombreux moments de joie partagés avec eux.

Toute ma famille en Syrie et en République Tchèque, et plus particulièrement mes parents, ma sœur Moni et mon frère Dany qui m'ont toujours encouragé. Un grand merci à mes parents à qui je dois ce que je suis devenue.

Mon mari Rémy pour son grand amour.

Résumé

La construction de logiciels par coordination d'applications existantes et autonomes permet de réaliser des logiciels de grande taille, réutilisant le savoir-faire contenu dans ces applications. Les procédés, utilisés dans différents domaines et de différentes manières pour coordonner des applications ou des humains, se présentent comme étant de bons candidats pour réaliser ces coordinations. Cependant, la plupart des procédés sont rigides, fortement couplés, et se basent sur un grand nombre de concepts spécialisés. Ils sont aussi souvent réservés à une catégorie d'applications. Nous présentons les points forts et les points faibles des formalismes et moteurs de procédés actuels et nous en déduisons quels sont les besoins pour la coordination d'applications via un procédé.

Nous présentons dans cette thèse l'environnement réalisé dans notre équipe qui propose une architecture des logiciels à trois couches : conceptuelle, médiateur, et outils. Le niveau conceptuel contient un meta-modèle qui ne définit que les concepts basiques des procédés. Nous définissons ensuite quatre classes d'extension de ce meta-modèle minimal. Notre environnement permet alors de construire et exécuter des logiciels basés sur des procédés de coordination extensibles et évolutifs.

Mots clés : Procédé, coordination, orchestration, ingénierie dirigé par les modèles, services web, programmation par aspects, fédération.

Abstract

Building software by a coordination of independent "components" allows to build large software while reusing the know how embedded in these components. The process technology, used for the coordination of human activities, is a good candidate for coordinating applications (instead of humans only). Nevertheless, current process formalism are rather primitive; they are rigid, strongly coupled, and propose a plethora of specialized concepts and mechanisms. We present the strong and weak points of process technology, and we deduce what are the needs for process driven applications.

We present the Mélusine environment built in our team, which proposes a three layers architecture : conceptual, mediator and tools. Conceptual layer contains a meta model defining only the basic concepts relevant in the domains of interest. We define then four different ways to extend this basic meta model. Our environment allows for building and executing applications based on extensible and evolutive coordination processes.

Keywords : Process, coordination, orchestration, model driven engineering, web services, aspect-oriented programming, federation.

Table des matières

Chapitre I.

Introduction

1. Le problème

Le but des "Procédés Logiciels" est de formaliser et d'automatiser la répétitivité de certaines suites de tâches du développement logiciel. Les procédés sont utilisés dans différents domaines et de différentes manières : dans le domaine du travail coopératif assisté par ordinateur, dans l'industrie (sous le nom de workflows), dans le domaine de la gestion de projet, et dans le domaine de l'orchestration de services web, apparu récemment. Nous avons étudié, dans cette thèse, ces différents types de procédés, et nous avons identifié les bénéfices qu'ils apportent et les défauts qu'ils contiennent. Nous présentons, dans cette thèse, ces différents types de procédés, en identifiant les caractéristiques de chacun de ces types, ainsi que leurs avantages et inconvénients.

La leçon majeure que nous avons apprise des différents domaines d'utilisation des procédés est qu'il est possible, et souvent souhaitable d'utiliser un procédé exécutable pour coordonner les logiciels ; nous appelons ces procédés des procédés de coordination. Nous avons identifié plusieurs défauts des procédés de coordination que nous résumons par les points suivants :

- La plupart de ces procédés sont rigides, et difficilement modifiables ;
- La plupart de ces procédés sont fortement couplés aux applications qu'ils coordonnent, la modification du lien entre le procédé et l'application qu'il coordonne est assez difficile ;
- Ces procédés sont souvent de bas niveau, et proche des langages de programmation ;
- La coordination est souvent réservée à des outils particuliers, ou à une technologie particulière, spécifique au domaine (par exemple, dans le domaine de l'orchestration, la coordination est souvent réservée aux services web) ;
- La plupart de ces procédés se basent sur un grand nombre de concepts spécialisés. Ces concepts sont souvent complexes, et leur nombre important surcharge le modèle de procédé et ajoute beaucoup de difficulté à la compréhension ;
- Dans certains domaines, comme le domaine de l'orchestration de service web, beaucoup de langages de coordination ont été proposés. Ce qui a donné naissance à une multitude de concepts qui se chevauchent.

D'autres défauts concernent le système de gestion de coordination, essentiellement :

- La performance est insuffisante ;
- La reprise sur pannes est manquante ;

- Il y a un manque d'outils d'analyse, de test, et de débogage ;
- Il y a un manque de prise en charge de l'interopérabilité entre le système et les outils qu'il coordonne.

Enfin, nous constatons souvent l'existence de plusieurs logiciels de gestion de coordination sur la même machine, concernant des domaines d'applications complémentaires, sans que ces logiciels communiquent afin de s'échanger leurs données, qui ont souvent des liens entre elles.

2. Objectifs de la thèse

Dans cette thèse, nous cherchons à résoudre les problèmes des procédés de coordination, en spécifiant les besoins de coordination, et en tentant d'éliminer les défauts mentionnés ci-dessus. Néanmoins, certains des besoins que nous avons identifiés sont assez contradictoires. Ainsi, le formalisme de description de procédé doit permettre l'introduction de nouveau concepts, pour permettre l'évolution et l'amélioration du procédé ; mais par ailleurs, ce même formalisme doit être simple, compréhensible, afin de permettre à ses utilisateurs de le manipuler sans difficultés. Mais en facilitant l'introduction de nouveaux concepts dans le formalisme afin de le faire évoluer, il peut devenir rapidement très complexe et difficile à gérer.

Dans cette thèse, nous proposons un environnement, Mélusine, permettant de construire des logiciels ayant une architecture à trois couches : concepts, médiateur, et outils. Cette architecture nous permet de résoudre beaucoup de problèmes que nous trouvons dans les logiciels de coordination à l'aide de procédés, tels que les problèmes de substitution, la facilité de modification du procédé, et l'utilisation d'un formalisme de haut niveau pour décrire la logique de l'application. Nous présentons les choix que nous avons pris pour définir cette architecture, et pour réaliser notre environnement.

L'approche que nous utilisons dans notre environnement consiste à se baser, au niveau conceptuel, sur un meta-modèle minimal, ne contenant que les concepts basiques. Ce meta-modèle minimal peut être étendu, afin de faire évoluer cet environnement. Nous définissons quatre classes d'évolutivité permettant d'étendre le logiciel construit à l'aide de notre environnement : l'extension sémantique, l'extension de méta-modèle, le raffinement, et la composition. Ces quatre formes d'évolutivité peuvent être combinées selon les besoins.

Dans ces quatre formes, les nouveaux concepts ajoutés pour un besoin particulier, ne se mélangent pas avec les concepts basiques de notre méta-modèle minimal. Cette séparation permet la réutilisation du logiciel de base et facilite la compréhension des concepts, car le concepteur du logiciel ne manipule qu'un nombre minimal de concepts, séparés par domaine d'application. L'implémentation de ces nouveaux concepts est également séparée, dans l'architecture que nous proposons, de l'implémentation de notre logiciel. Ce qui permet de bien structurer notre application globale.

Les quatre formes d'évolutivité proposés ne modifient pas le logiciel que nous souhaitons faire évoluer. Les extensions peuvent être activées ou non au moment de l'exécution, ce qui permet d'exécuter à volonté notre logiciel dans son état d'origine ou avec certaines ou toutes ses extensions.

L'architecture que nous proposons pour l'évolution des logiciels est une architecture modulaire, du fait de la séparation entre les différents domaines d'application et d'extension. Cette modularité offre beaucoup de flexibilité, simplifie la maintenance, et facilite la réutilisation.

Notre approche proposant ces quatre formes d'évolutivité s'adresse à un très large spectre d'applications, mais surtout à celles qui couvrent différents domaines d'application. Elle est applicable à tout logiciel, même si l'architecture de ce logiciel n'est pas tout à fait conforme à l'architecture que propose notre plate-forme Mélusine. La seule condition pour pouvoir appliquer nos quatre formes d'évolutivité à un logiciel est que les concepts représentant la logique de ce logiciel soient séparés de leur implémentation, et réifies à l'exécution.

L'environnement que nous proposons permet de gérer la construction et l'exécution de procédés génériques, utilisés pour la coordination de tâches, et couvrant plusieurs domaines d'utilisation. Ainsi, en utilisant notre approche, et à l'aide de notre environnement, nous pouvons construire et exécuter des logiciels, basés sur des procédés extensibles et évolutifs de coordination. Ces extensions sont gérées de manière séparée du meta-modèle de base, ce qui permet d'avoir un meta-modèle compréhensible, et simple à gérer.

3. Plan de la thèse

Ce document de thèse est organisé en deux parties :

- La première partie présente une étude des différents domaines utilisant les procédés. Ceci en identifiant les caractéristiques de chacun de ces types, ainsi que leurs avantages et inconvénients.
  - Le chapitre 2 présente les domaines du procédé logiciel, du travail coopératif assisté par ordinateur, des workflows, et de la gestion de projet,
  - Le chapitre 3 présente l'orchestration de services web, ce thème étant central dans cette thèse, est présenté dans un chapitre à part.
- La deuxième partie présente notre proposition.
  - Le chapitre 4 présente notre environnement Mélusine, et l'architecture à trois couches que propose cet environnement,
  - Le chapitre 5 présente les quatre classes d'évolutivité permettant d'étendre le logiciel construit à l'aide de notre environnement : l'extension sémantique, l'extension de méta-modèle, le raffinement, et la composition.

Finalement, le chapitre 6 présente la conclusion de ce document, ainsi que les travaux futurs.

Chapitre II.

Etat de l’art sur les procédés

1. Définitions et classification des procédés

Les procédés sont définis comme étant une suite d'étapes réalisées dans un but donné : *"a sequence of steps performed for a given purpose"* [IEEE-STD]. Ils ont donc toujours existé.

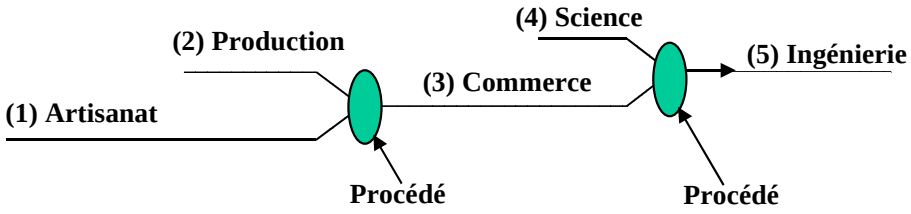


Figure 1 : l’apparition des procédés dans le monde [Shaw 90]

Les procédés sont apparus dans le monde de l’informatique dans les années 80, lorsque les informaticiens se sont aperçus de la répétitivité de certaines suites de tâches, et ont cherché à les formaliser et/ou les automatiser. Les informaticiens se sont intéressés en premier à la formalisation des tâches du développement logiciel, ce qu’ils ont appelé le "Procédé Logiciel". L’utilisation des procédés c’est ensuite étendu à d’autres domaines. Elle a permis de faire évoluer ces domaines, en introduisant cette nouvelle technique. Dans la Figure 2, je donne une liste de domaines utilisant les procédés, et les dates du début de l’utilisation de procédé pour chacun d’entre eux.

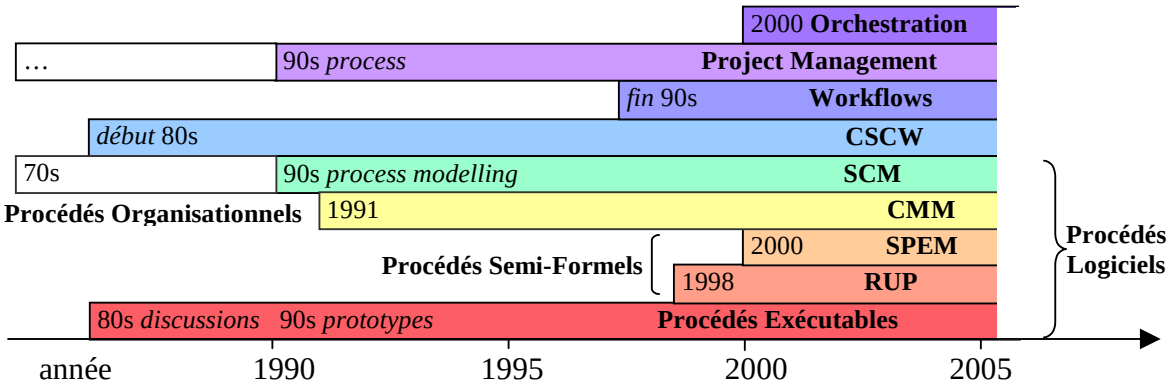


Figure 2 : l’histoire des procédés

Les procédés informatiques sont utilisés de manière différente dans les différents domaines :

- Les "Procédés Logiciels" (Software Process) servent à gérer et assister le développement logiciel. Il existe plusieurs sous-classes de Procédés Logiciels :
  - Les Procédés Exécutables, ils assistent le développement logiciel de manière exécutable,
  - Les Procédés Semi-Formels, nous mentionnons dans cette catégorie :
    - Le "Processus Unifié" (RUP : Rational Unified Process). Dans RUP, les procédés servent à modéliser la façon dont le logiciel va être développé,
    - GQM (Goal/Question/Metric),
    - QIP (Quality Improvement Paradigm),
    - TSP (Team Software Process),
    - PSP (Personal Software Processs),
  - Les Procédés Organisationnels, comme "CMM" (Capability Maturity Model). CMM a décrit les éléments clés d'un processus logiciel efficace,
  - Les Procédés de Gestion de Configuration (SCM : Software Configuration Management). Dans ce domaine, les procédés sont utilisés pour contrôler et assister l’évolution du logiciel,
  - Les Procédés Logiciels récents, comme "SPEM" (Software Process Engineering Metamodel). Dans SPEM, les procédés servent à décrire le développement logiciel (SPEM s’occupe de la description et non pas de l’exécution de procédé),
- Le "Travail Coopératif Assisté par Ordinateur" (CSCW : Computer Supported Co-operative Work). Dans le domaine du CSCW, les procédés sont utilisés pour coordonner les membres d’un groupe,
- Dans les "Workflows", les procédés servent à :
  - Automatiser, partiellement ou totalement, les activités du travail, surtout les activités de bureautique (office automation), et ceci pour améliorer l’efficacité,
  - Gérer l’intégration et l’interopérabilité des systèmes d’information HAD (Hétérogènes, Autonomes, et/ou Distribués),
- Dans le domaine de la "Gestion de Projet" (Project Management), les procédés sont utilisés pour gérer les ressources et le cycle de vie du projet,
- Et finalement, dans "l’Orchestration et la Chorégraphie de services web", les procédés servent à la coordination des services web.

Dans ce chapitre, nous allons présenter ces différentes classes de procédé. Pour cela, il est important de définir ce qu’est un modèle et un méta-modèle de procédé. Pour ces définitions, nous nous référons au domaine de l’architecture dirigée par des modèles (MDA : Model Driven Architecture [BB 02]) proposé par l’OMG (Object Management Group [OMG]).

Modèle et Méta-Modèle [1]

Afin d'organiser et de structurer les modèles, l'OMG a défini une architecture appelée : "Architecture à quatre niveaux". Ces quatre niveaux sont (Figure 3) :

- **Le niveau M0 :** C'est le niveau des données réelles. Il est composé des informations que l'on souhaite modéliser. Ce niveau est souvent considéré comme étant le monde réel ;
- **Le niveau M1 :** Lorsque l'on veut décrire les informations appartenant à M0, le MDA considère que l'on fait un modèle appartenant au niveau M1. Un modèle de procédé appartient donc au niveau M1. De même, tout modèle de niveau M1 est exprimé dans un langage dont la définition est fournie explicitement au niveau M2 ;
- **Le niveau M2 :** Ce niveau est composé de langages de définition des modèles, appelés aussi méta-modèles. Le méta-modèle de procédé appartient au niveau M2, et définit la structure interne des modèles de procédé ;
- **Le niveau M3 :** Ce niveau contient le MOF (Meta-Object Facility), le langage unique de définition des méta-modèles, aussi appelé le méta-méta-modèle. Le MOF définit la structure de tous les méta-modèles qui se trouvent au niveau M2.

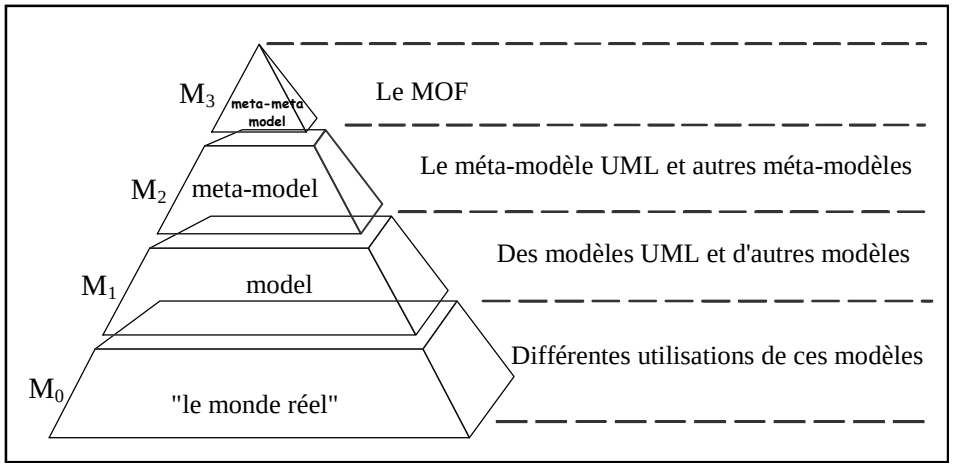


Figure 3 : Architecture à 4 niveaux de l'OMG

2. Les Procédés Logiciels (Software Process)

Dans cette section, nous allons parler des différentes classes de procédés logiciels :

- Les Procédés Exécutables ;
- Les Procédés Semi-Formels ;
- Les Procédés Organisationnels ;
- Les Procédés de Gestion de Configuration ;
- Les Procédés Logiciels récents.

[1] Extraits de [BB 02]

2.1. Les Procédés Logiciels Exécutables

Le Procédé Logiciel Exécutable définit la manière dont le développement logiciel est organisé, géré, mesuré, assisté, et amélioré (indépendamment du type de support technologique choisi pour le développement) [Der 99].

Les procédés logiciels peuvent être exprimés sous forme de modèle de procédé, décrit dans un Langage de Modélisation de Procédé (PML : Process Modelling Language). Un modèle de procédé est une représentation des activités du monde réel. Le modèle de procédé logiciel est développé, analysé, raffiné, transformé, et/ou exécuté conformément au méta-modèle du procédé [Der 99 - Chapitre 3 – paragraphe 3.2].

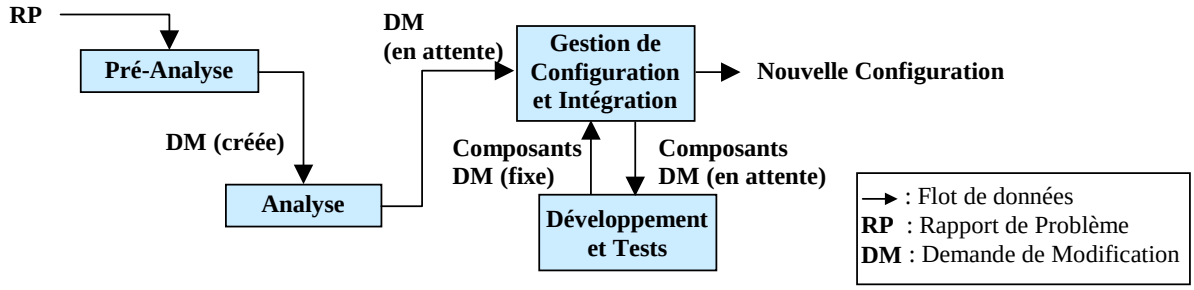


Figure 4 : un modèle de procédé pour la modification de logiciel [Der 99]

Dans cette section, nous allons présenter les éléments du méta-modèle de procédé logiciel, et les catégories des Langages de Modélisation de Procédé (PMLs).

2.1.1. Eléments du méta-modèle de Procédé Logiciel [2]

Les éléments du méta-modèle de procédés logiciels se décomposent en éléments basiques et éléments secondaires. Les éléments basiques sont les suivants :

Activité

Une activité est une tâche pendant laquelle des opérations sur le logiciel à développer sont accomplies. Elle est souvent associée à une ou des personnes responsables de cette activité, et à des outils de production.
Une activité peut être décomposée en d'autres activités, formant ainsi plusieurs niveaux d'abstraction. L'activité peut être concurrente et coopérative, déterministe ou non-déterministe.
Le modèle de procédé logiciel comprend des activités de développement du logiciel et de maintenance, des activités de gestion de projet et d'assurance qualité, et des activités de méta-procédé. Les activités peuvent avoir différents niveaux de granularité, et sont généralement associés à des rôles pouvant être entrepris par certaines catégories d'utilisateurs et/ou d'outils. Les produits sont les données d'entrée et de sortie des activités.

Produit

[2] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.2.1 Process Elements"

Un produit est souvent un artefact (persistant et versionné), pouvant être simple ou composite, formant les données d'entrée et de sortie des activités. Les produits peuvent être des parties du logiciel à développer, et des documents associés (documents de conception, documentation de l'utilisateur, données de test).

Rôle

Un rôle décrit les droits et les responsabilités d'un humain. Un humain joue un rôle dans une activité (ou plusieurs rôles dans des activités différentes).

Humain

Les humains sont des agents (ou des développeurs) du modèle de procédé logiciel, que l'on peut organiser dans des groupes. Un rôle est attribué aux humains ayant les compétences et les responsabilités nécessaires pour jouer ce rôle. Un humain peu avoir plusieurs rôles, il peut également être membre de plusieurs groupes (ces groupes peuvent aussi être imbriqués).

Outil

Les outils sont des systèmes qui assistent la production de logiciel. Il existe deux sortes d'outils : les outils interactifs (éditeurs textuels, outils graphiques comme CASE…), et les outils simplement exécutables sans interaction (compilateurs, analyseur grammatical…).

Support d'évolution (Directions)

Le support d'évolution aide à gérer l'évolution du procédé logiciel (sa modification), à travers les Directions (politiques, règles, et procédures). L'évolution du procédé logiciel est un besoin essentiel, à cause de sa nature orientée-humain. Une façon de supporter l'évolution est d'utiliser un méta-modèle de procédé offrant une assistance du point de vue conceptuel, pour les changements statiques ou dynamiques du modèle de procédé. Une assistance technique serait également importante pour supporter l'évolution.

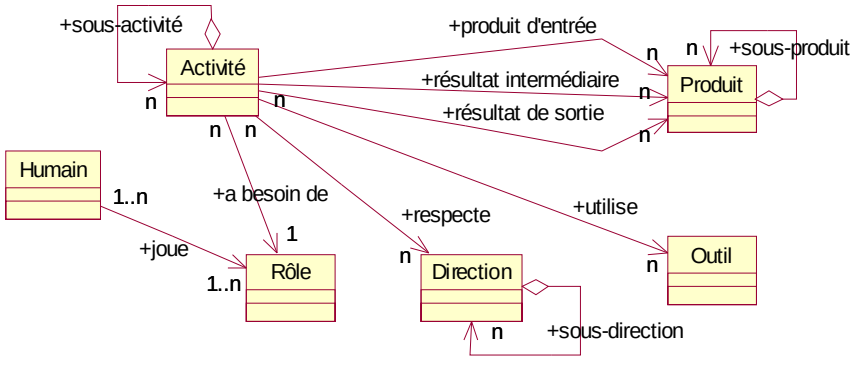


Figure 5 : les éléments basiques de méta-modèle de procédé logiciel [3]

[3] Repris de [Der 99] Chapitre 1 : "The Software Process: Modelling and Technology" - "1.5. Process Modelling"

[Der 99] a identifié les éléments secondaires suivants : Projet/organisation, Espace de travail, Vue utilisateur, Modèle de coopération, Modèle de versionnement/transaction, et Modèle de qualité/performance.

Nous reprenons ces concepts du meta-modèle de procédé dans le Chapitre IV, lorsque nous parlons de notre méta-modèle de procédé, utilisé dans la plate-forme Mélusine, pour construire des logiciels de coordination d'outils à l'aide d'un procédé.

2.1.2. Catégories des Langages de Modélisation de Procédé (PMLs) [4]

Les Langages de Modélisation de Procédé (PMLs) et leurs Environnements de Génie Logiciel Sensible au Procédé (PSEEs) peuvent être classifiés en trois catégories :

- Les PMLs centrés Produits : cette catégorie contient le système ADELE et le système EPOS. Les PMLs associés sont souvent OO (Orientés Objet) ou EER (Extended Entity Relationship) ;
- Les PMLs centrés Activités : ces PMLs sont conçus pour fournir un support pour la description d'activités. Comme exemple de ces PMLs nous avons :
  - MARVEL, MERLIN, OIKOS, Tempo (basés sur des règles et des déclencheurs),
  - Process Weaver, SPADE, MELMAC, EPOS (basés sur un réseau d'activités),
  - APPL/A (fournissant un PML impératif),
  - APEL (qui est une évolution du PML centré produits ADELE),
- Les PMLs centrés Rôles : ces PMLs sont centrés autour des rôles d'humains, et leurs interactions. Un exemple de ces PMLs : PWI, et les systèmes de groupware ou de speech-act.

1) Les PMLs centrés Produits

Dans cette section nous présentons deux systèmes de PML centré Produits : le système ADELE et le système EPOS.

ADELE [5]

ADELE a été une des premières tentatives en Modélisation de Procédé, il est basé sur les langages orientés objet, la modélisation de bases de données, et les mécanismes de déclencheurs (trigger). Il propose deux Langages de Modélisation de Procédé (PML). Le langage original est un langage de base de données orienté objet, étendu par des relations et des déclencheurs. Les documents, les outils, et les contextes de travail sont les éléments basiques de ce langage exécutable. Les activités peuvent être exprimées à l'aide de déclencheurs, formés de règles du style ECA (événement/condition/action), associés aux opérations de la base de données. Le premier langage a été jugé efficace pour l'implémentation et l'exécution de procédé, mais pas pour la spécification et le contrôle, car ce

[4] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.6 Possible Groups of PMLs and PSEEs"

[5] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.4.7 ADELE-TEMPO"

formalisme est difficile à comprendre pour les humains, et l'exécution des déclencheurs est difficile à contrôler.

EPOS SPELL [6]

EPOS SPELL, le Langage de Modélisation de Procédé de EPOS, contient un noyau extensible de types d'entités et de types de relations prédéfinies ("tâche", "donnée", "outil", et "rôle") pouvant être spécialisés par héritage.
Le type "tâche" contient :

- les PRE- et POST- conditions, qui sont des expressions logiques de premier niveau (comme dans l'intelligence artificielle) ;
- le FORMEL, qui est constitué des déclarations de paramètres d'entrée et de sortie (comme dans les diagrammes de flot de données) ;
- la DECOMPOSITION, qui définie les décompositions des tâches (comme dans la décomposition fonctionnelle).

SPELL est implémenté au-dessus d'une base de données orientée objet, fournissant un support pour la persistance, le versionnement, et les transactions. C'est un langage réflexif, fournissant un support pour l'évolution et la définition explicite de méta-modèle de procédé.

SPELL est basé sur des langages orientés objet, les bases de données, les systèmes réflexifs, la planification en intelligence artificielle, et les systèmes de flot de données. Il contient les types SPELL, qui peuvent être définis en une notation textuelle basée sur Prolog, convenable pour l'implémentation, mais pas pour les phases de plus haut niveau comme l'obtention du procédé et les spécifications des besoins.

2) Les PMLs centrés Activités

Dans cette section, nous présentons des exemples de PMLs centrés Activités. Nous présentons :

- MARVEL un exemple de PML basés sur des règles et des déclencheurs ;
- SPADE un exemple de PML basés sur un réseau d'activités ;
- APPL/A fournissant un PML impératif ;
- APEL qui est une évolution du PML centré produits ADELE, présenté précédemment.

MARVEL [7]

La version 3.1 de Marvel contient un langage de modélisation nommé MSL, contenant trois concepts principaux : les classes, les règles, et les enveloppes d'outils.

- Les classes contiennent des attributs, qui sont soit des attributs typés ou des références vers d'autres classes ;
- Les règles déclarent les pré- et post-conditions et le code.

[6] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.4.2 EPOS SPELL"

[7] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.5.2 MARVEL "

Le Langage de Structures d'Activités (ASL Activity Structures Language) [KPB 94] a été implémenté dans Marvel. Ce langage, ASL, est une variante des expressions de contraintes de Riddle [Rid 91], permettant la modélisation de la topologie du modèle de procédé, à l'aide d'expressions régulières étendue par des opérateurs concurrentiels. Les parties écrites en ASL peuvent être traduites vers du MSL.

SPADE [8]

Le Langage de Modélisation de Procédé de SPADE s'appelle SLANG. Il est constitué de deux niveaux de langages :

- le noyau de SLANG, qui est un langage formel basé sur les réseaux de Pétri (pour la définition comportementale), et sur l'orienté objet (pour la définition de structure statique) ;
- SLANG enrichis le noyau par des améliorations syntaxiques spécifiques à la modélisation de procédé, et avec des types spéciaux pour l'évolution de la modélisation de procédé.

SLANG est intégré avec la base de données orienté objet O2, qui agit comme un répertoire pour le modèle de procédé et les produits du procédé.

Il est possible de modéliser les politiques de coopération, comme faisant part du modèle de procédé. On obtient ceci à travers des mécanismes d'interaction d'outils de procédé, incorporé dans des concepts de langage spécial ("black transitions" et "user places"). Les outils peuvent être intégrés directement dans l'environnement de SPADE, ou bien à travers des produits commerciaux d'intégration d'outils, comme par exemple DEC FUSE et SUN ToolTalk.

APPL/A [9]

APPL/A a été développé dans le contexte du projet ARCADIA, c'est un langage de programmation de procédé. APPL/A est une extension de ADA, ajoutant la persistance, la gestion de relations, les transactions, et les déclencheurs (triggers). Ainsi, le langage est essentiellement procédural, mais les contraintes de cohérence peuvent être exprimées comme des prédicats sur les relations.

WEADELE (APEL V1) [10]

A la suite d'ADELE, dans le cadre du projet PERFECT Esprit, le langage APEL (Abstract Process Engine Language) a été développé. APEL est un langage graphique de haut niveau, conçu à la fois pour capturer, comprendre, et exécuter le procédé. Le modèle de procédé décrit en APEL est compilé vers un moteur de procédé abstrait, construit à partir de deux moteurs de procédé basiques commerciaux : "Adele" et "Process Weaver", intéropérant d'égal à égal (peer to peer). Dans APEL, le procédé est représenté utilisant différents aspects : le flot de contrôle, le flot de données, la description de données (à la manière OMT /Modélisation et

[8] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.4.8 SPADE "

[9] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.5.1 APPL/A"

[10] Extrait de [Der 99] Chapitre 3 : "Process Modelling Languages" - "3.4.7 ADELE-TEMPO"

Conception Orientées Objet [RBPEL 91]/), les espaces de travail et la coopération, les rôles (d'utilisateurs), et le diagramme de transition d'états.

Les aspects sont interdépendants et la cohérence est imposée, mais il n'y a pas d'aspect principal. L'utilisateur peut décrire le procédé en utilisant tous les aspects ou une partie d'eux. Chaque description est un niveau d'abstraction. Chaque niveau peut être raffiné récursivement.

3) Les PMLs centrés Rôles

Dans cette section, nous présentons comme exemple de PMLs centrés rôles, ProcessWise Integrator (PWI).
Les systèmes de "groupware" forment aussi un exemple de PMLs centrés rôles. Nous parlons des groupwares dans la section 3 concernant le Travail Coopératif Assisté par Ordinateur (CSCW : Computer Supported Co-operative Work).

PWI (ProcessWise Integrator) [11]

L'intégrateur de ProcessWise permet de modéliser le procédé, de l'intégrer, et d'exécuter le système basé sur ce procédé.

Le Langage de Modélisation de Procédé a été conçu comme un langage basé sur des classes. La hiérarchie de classes contient trois sortes de classes : Entités, Actions, et Rôles.

- Les définitions de la classe Entité créent les types d'enregistrement ;
- Les définitions de la classe Action introduisent les procédures ;
- Et les définitions de la classe Rôle agissent comme des schémas pour la création de thread d'exécution à venir.

Une définition de classe de rôle définit les valeurs de données, ainsi que le comportement de ses instances à travers un thread de procédé simple. Les propriétés des données sont définies dans la catégorie de ressources, et ces valeurs de données sont possédées exclusivement par ce rôle. Un rôle peut communiquer uniquement avec un autre rôle, en lui envoyant un message. Ce mécanisme de communication est asynchrone.

Le corps du rôle est également divisé en ce que l'on appelle des propriétés d'actions, ces propriétés décrivent le comportement du rôle.

4) Comparaison entre ces différentes catégories

Il y a eu donc trois classes principales de Langages de Modélisation de Procédé, suivant l'élément central du modèle de procédé. Ainsi, il y a eu des PMLs centrés Produits, Activités, et Rôles.

[11] Extrait de [BGR 94]

Les PMLs centrés Produits se concentrent surtout sur les données échangées, et ont développé d'importantes propriétés concernant ces données (données orientées objet, transactions, persistance, versionnement…). Malgré les difficultés qu'ont rencontré les PMLs centrés Produits, ils ont quand même eu beaucoup de succès, surtout dans le domaine de la gestion de configuration. Cette classe de PML a été fortement utilisée dans l'industrie, ce qui n'a pas été le cas des deux autres classes de PML.

Dans les PMLs centrés Rôles, l'accent est mis sur les rôles et sur la communication et la collaboration entre rôles. Cette classe de PML n'a pas eu beaucoup de succès dans le domaine des procédés logiciels. Par contre, elle a été utilisée dans le domaine du travail coopératif, dont nous allons parler dans la section 3.

Dans les PMLs centrés Activités, ce sont les activités, représentant les tâches à réaliser, qui forment le concept central. Cette classe de PMLs n'a eu du succès que dans le domaine de la recherche. Dans les années 90, beaucoup de travaux de recherches ont été effectués sur cette classe, c'était d'ailleurs la classe favorite des chercheurs. Ces travaux de recherches ont permis de bien spécifier les concepts des PMLs centrés Activités, mais ils n'ont donné lieu – à ma connaissance - à aucune implémentation commercialisée. Même les quelques tentatives de commercialisation qui ont eu lieu n'ont finalement pas eu de succès. Parmi ces tentatives, nous mentionnons Endeavors [BT 96] et FunSoft [GJ 92] qui ont été poussé dans le cadre commercial, ainsi que SPADE [BBFGL 94], Marvel [Bar 92, Kai 98], Merlin [JPSW 94], et Oz [BK 94] qui ont été distribué en logiciels libres (open source).

Du coté industriel, la construction de workflows a commencé vers la fin des années 90, et ceci indépendamment de tous les travaux de recherche qui ont été fait sur les procédés. Les workflows sont des procédés exécutables, centré Activités, et très simples, souvent restreins à des enchaînements simples d'activités (les workflows seront présentés dans la section 4). Ces workflows ont eu un certain succès dans l'industrie. Les industriels ont développé et utilisé des workflows pour gérer leurs travaux. Mais ce succès n'a pas été le fruit des travaux de recherche faites sur les PMLs centrés Activités. Il a été surtout la réponse aux besoins des entreprises à organiser leur travail de manière coordonnée à l'aide de workflows, formalisant l'enchaînement des tâches à réaliser dans l'entreprise.

2.1.3. Notre PML de Procédé Logiciel Exécutable : APEL V4/V5

Dans notre équipe, un PML de Procédé Logiciel Exécutable a été développé : "APEL" (Abstract Process Engine Language) [EDA 98, ECB 98, ED 96, EAD 99]. APEL est un PML de procédé logiciel centré activités, il est l'évolution du système ADELE dont nous avons parlé précédemment (voir ADELE et WEADELE (APEL V1) dans la section 2.1.2).

Le système ADELE (1988) permet de modéliser des procédés de gestion de configuration. Ces procédés s'exécutent en utilisant un système ECA (événement/condition/action), par des mécanismes de déclencheurs (trigger) [BE 87]. Le travail de notre équipe autour d'APEL a commencé vers l'année 1988, dans l'objectif de fournir un environnement de support complet

pour la gestion des procédés logiciels [BEM 91]. Quatre versions successives ont été construites entre 1992 et 1997.

Dans la première version, APEL V1 (ou WEADELE), l'implémentation était complètement basée sur ADELE. Le procédé était complètement traduit en terme de triggers, implémentant un système ECA. A partir de la deuxième version, l'objectif fut de construire un formalisme de haut niveau pour cacher les triggers. Cette deuxième version d'APEL a utilisé ADELE et Process Weaver. Le modèle de procédé décrit en APEL est compilé vers un moteur de procédé abstrait, construit à partir de deux moteurs de procédé basiques : ADELE et Process Weaver. L'assemblage des ces deux moteur s'étant révélé problématique, APEL V3 s'est alors focalisé sur la définition d'un formalisme de haut niveau exécuté par un moteur unique.

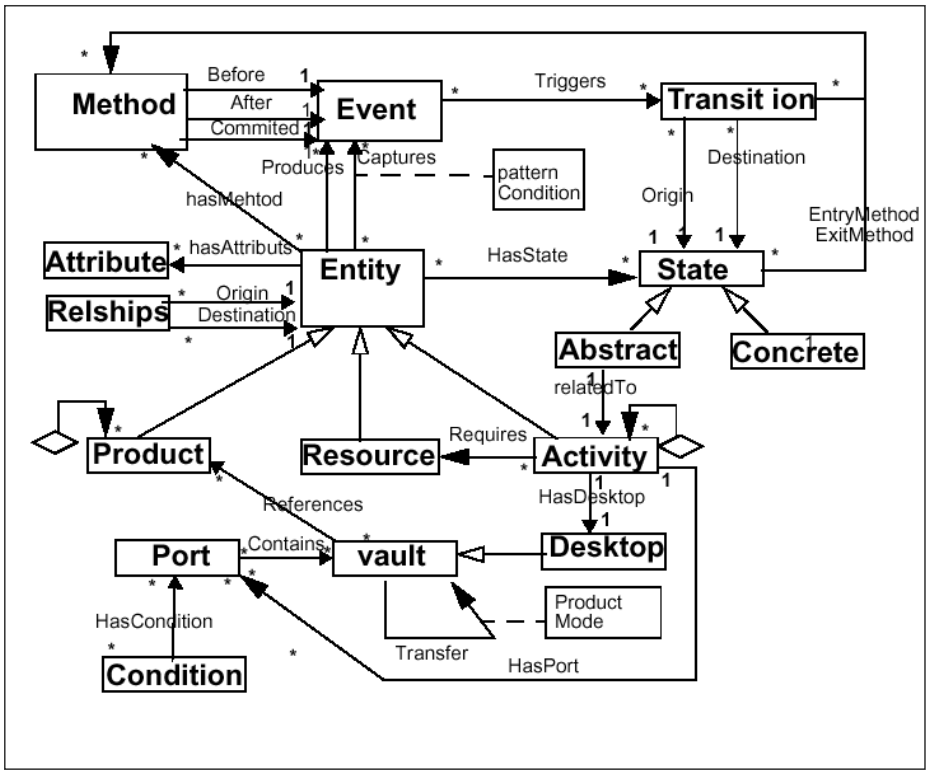


Figure 6 : le meta-modèle de APEL V4

La quatrième version, APEL V4 (Figure 6), forme un environnement de support des procédés logiciels très riche en concepts, il contient un formalisme graphique de haut niveau pour modéliser les procédés, un système d'exécution de procédés de haut niveau, et un support à l'évolution dynamique de ces procédés logiciels. Néanmoins, la richesse de concepts que nous trouvons dans APEL V4 l'a rendu très complexe, et difficile à manipuler. En plus, APEL V4 est un système monolithique, ce qui rend son évolution difficile.

Au début des années 2000, notre objectif fut de casser le monolithisme d'APEL V4 afin de mieux le faire évoluer. Nous avons eu comme but également d'alléger le meta-modèle de APEL, afin de le rendre plus simple à utiliser. Ceci, car nous avons remarqué que beaucoup de concepts de APEL V4 n'étaient utilisés que rarement.

Pour cela, nous avons construit la version APEL V5 [Vil 03, Le 04], qui est un noyau minimal, ne contenant que les concepts principaux, formant les éléments de base : les activités, les produits, leur types, les rôles, les ressources, les flots de données, et les ports (la Figure 7 illustre le méta modèle d'APEL V5). Nous détaillons les concepts du meta-modèle de APEL V5 dans la section 2.1.1. du chapitre 4.

Ce noyau minimal peut être étendu, afin de lui ajouter de nouveaux concepts et de nouvelles fonctionnalités quand il y a besoin. Nous présentons dans cette thèse différentes façons d'étendre et de faire évoluer notre environnement de procédé. Le chapitre 4 détaille ces différentes formes d'extensions.

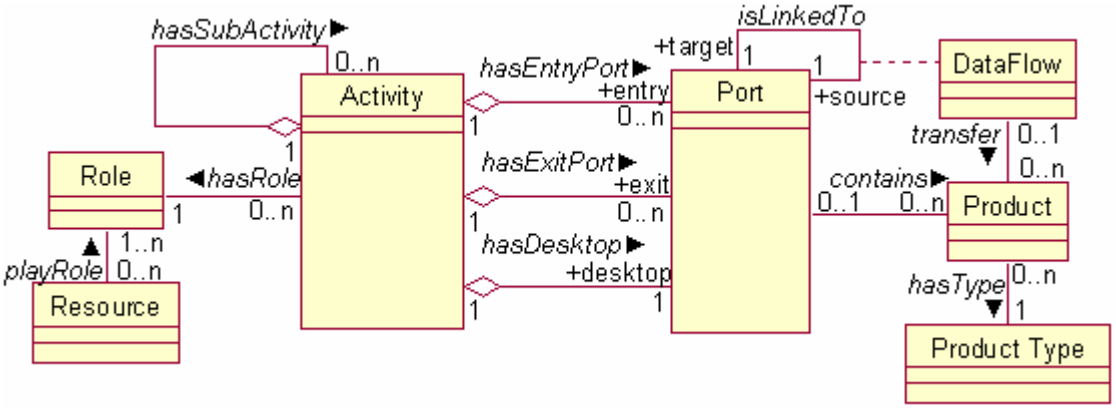


Figure 7 : le meta-modèle de APEL V5

2.1.4. Synthèse sur les Procédés Logiciels Exécutables

Les procédés logiciels exécutables ont été utilisés pour gérer et assister le développement logiciel. Il y a eu trois catégories de procédés logiciels exécutables : les PMLs centrés Activités, centrés Produits, et centrés Rôles, cela suivant le concept central sur lequel l'accent a été mis.

L'approche centrée Activités a été l'approche la plus prometteuse, qui a concentré l'essentiel des travaux. Ces travaux on permis d'explorer et de mettre en place le vocabulaire et les ontologies spécifiques à cette classe de procédés, ce qui a permis de mieux comprendre le domaine de procédés centré Activités. Cependant, aucune implémentation commerciale réussie n'a vu le jour.

Nous pouvons expliquer cela par le fait que les ambitions étaient très élevées, et que la technologie des procédés centrés Activités est arrivée en avance par rapport aux besoins industriels. Ceci est confirmé par le fait que les entreprises ont commencé, vers la fin des années 90, à formaliser leur fonctionnement, en mettant en place des workflows. Les workflows, comme nous allons le présenter dans la section 4, sont des procédés exécutables, centré Activités, souvent restreints à des enchaînements simples d'activités.

Dans notre équipe, nous avons construit un PML centré Activités de procédés logiciels exécutables : APEL. Dans notre approche, nous nous basons sur un meta-modèle de procédé minimal, ne contenant que les concepts basiques. Ce meta-modèle minimal peut être étendu, afin de faire évoluer l'environnement de procédé.

2.2. Les Procédés Semi-Formels

Dans cette classe de procédés logiciels, on trouve :

- Le Processus Unifié (RUP : Rational Unified Process).
- GQM (Goal/Question/Metric).
- QIP (Quality Improvement Paradigm).
- TSP (Team Software Process).
- PSP (Personal Software Processs).

2.2.1. Le Processus Unifié (RUP Rational Unified Process) [12]

RUP est un modèle de procédé logiciel générique, où le développement logiciel est vu comme une activité itérative de quatre phases : l'initialisation, l'élaboration, la construction, et la transition. L'initialisation établit une analyse de cas pour le système, l'élaboration définit l'architecture, la construction implémente le système, et la transition déploie le système dans l'environnement du client [Som 04].

1) Les deux dimensions du Procédé Unifié (RUP)

La Figure 8 illustre l'architecture du RUP dans son ensemble. Dans cette architecture, le procédé a deux dimensions :

- L'axe horizontal représente le temps, et montre les phases du cycle de vie du procédé. Cette dimension représente l'aspect dynamique du procédé lors de l'exécution. Il est exprimé en terme de cycles, phases, itérations et jalons.

- L'axe vertical représente les disciplines du procédé, regroupant les activités par leur nature. Cette dimension représente l'aspect statique du procédé : comment est-il décrit en termes de composants de procédé, d'activités, de workflow, d'artefacts et de travailleurs.

[12] Extraits de [Kru 00] et de [Roq 02]

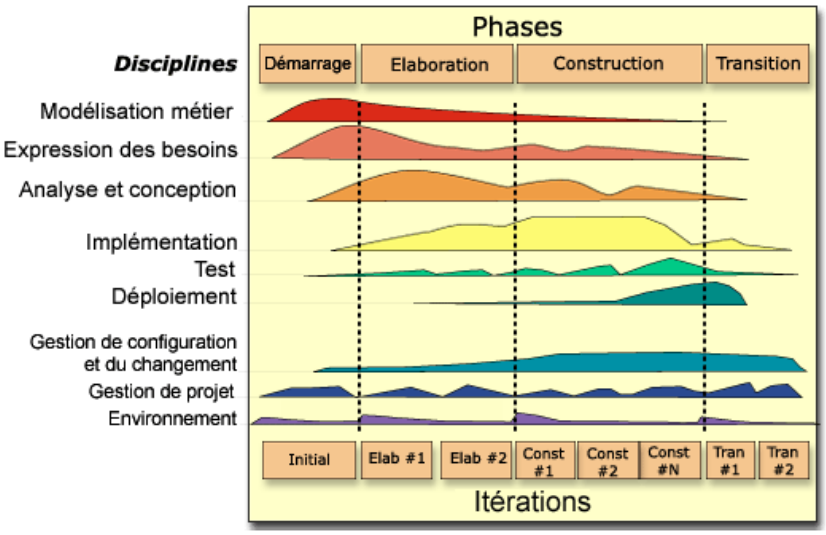


Figure 8 : les deux dimensions du Procédé Unifié (RUP)

2) Les meilleures pratiques de développement sont assistées par RUP

RUP assiste les six meilleures pratiques de développement logiciels qui sont :

- Le développement itératif ;
- La gestion des besoins ;
- L'architecture et l'utilisation de composants ;
- La modélisation et UML ;
- La qualité des procédés et des produits ;
- La gestion de configuration et de changements.

3) Trois caractéristiques additionnelles de RUP

En plus de ces six meilleures pratiques de développement, RUP assiste trois activités additionnelles importantes :

- Le développement guidé par les cas d'utilisation ;
- La configuration de procédés ;
- La mise à disposition d'outils de support.

4) Synthèse

RUP est une méthode de développement logiciel qui sert à modéliser la façon dont le logiciel va être développé, en définissant les risques dès le début et en itérant dans chaque phase pour corriger les incohérences.

RUP est un procédé logiciel semi-formel qui a eu et a toujours beaucoup de succès. De nombreux outils ont été implémentés afin d'assister les programmeurs utilisant RUP.

2.2.2. Objectif/Question/Métrique (GQM Goal/Question/Metric)[13]

La méthode GQM est utilisée pour définir des mesures de projet logiciel, de procédé, et de produit de façon à ce que :

- Les métriques résultantes soient adaptées à l'organisation et ses objectifs ;
- Les données des métriques résultantes jouent un rôle constructif et instructif dans l'organisation ;
- Les métriques et leurs interprétations reflètent les valeurs et les points de vues des différents groupes affectés (ex. développeurs, utilisateurs, opérateurs).

GQM définit un modèle de mesure à trois niveaux :

- Le niveau conceptuel (objectif) : un objectif est défini pour un élément, pour diverses raisons, en respectant différents modèles de qualité, de différents points de vue, et relatif à un environnement particulier ;
- Le niveau opérationnel (question) : un ensemble de questions est utilisé pour définir des modèles de l'élément étudié puis se focalise sur cet élément pour caractériser l'évaluation ou le succès d'un objectif spécifique ;
- Le niveau quantitatif (métrique) : un ensemble de métriques, basées sur les modèles, est associé à chaque question afin d'y répondre de manière quantifiable.

Bien que originalement GQM ait été utilisé pour définir et évaluer un projet particulier dans un environnement particulier, cette méthode peut également être utilisée pour le contrôle et l'amélioration d'un projet simple à l'intérieur d'une organisation contenant plusieurs projets.

2.2.3. Quality Improvement Paradigm (QIP)[14]

Le modèle QIP propose d'assister l'amélioration continue du procédé et l'ingénierie des procédés de développement, ainsi que d'aider le renouvellement technologique.

QIP est basé sur le principe que la discipline du logiciel est, par nature, évolutive et expérimentale. Le développement logiciel est basé sur les humains. Il contient très peu de répétitions, ce qui rend l'utilisation des contrôles statistiques comme celles utilisés dans l'industrie extrêmement difficile et douteuse. Les développeurs de QIP prennent clairement une approche différente des auteurs de CMM (présenté dans la suite de ce document), et un certain nombre d'autres modèles basés sur l'idée d'un contrôle statistique de procédés.

Le cycle de QIP contient deux circuits fermés en forme de boucle : la boucle organisationnelle (plus large) et la boucle du projet (plus petite).

- la boucle du projet produit des retours (feedback) pendant la phase d'exécution afin de prévenir et résoudre les problèmes, superviser et assister le projet et pour redéfinir les procédés choisis afin qu'ils respectent les objectifs définis ;

[13] Extrait de [SEL GQM]

[14] Extrait de [Kin 00]

- la boucle organisationnelle produit, après l'achèvement du projet, des retours vers l'organisation. L'objectif de ces retours organisationnels est d'analyser les similarités et les divergences des données rassemblées par rapport aux expériences et aux modèles précédents.

QIP tente d'accumuler et de capitaliser au mieux de l'expérience au fil des projets de façon à ce que les projets futurs et en cous puissent en bénéficier.

La boucle organisationnelle représente l'apprentissage de l'organisation. Elle se divise en six phases :

- Caractériser et comprendre ;
- Fixer les objectifs ;
- Choisir les procédés, les méthodes, les techniques et les outils ;
- Exécuter les procédés (exécuter la boucle du projet) ;
- Analyser les résultats ;
- Assembler et stocker l'expérience.

La boucle du QIP peut être utilisée comme un outil pour en apprendre plus sur les l'expérience assemblée existante, ou bien pour créer un assemblage d'expériences complètement nouveau.

2.2.4. Team Software Process (TSP)[15]

TSP, accompagné du PSP dont nous allons parler dans la section suivante, aide l'ingénieur à :

- assurer la qualité des produits logiciels ;
- créer des produits logiciels sécurisés ;
- améliorer la gestion de procédé dans une organisation.

Les équipes d'ingénierie utilisent TSP pour appliquer les concepts d'équipe intégrée au développement de systèmes logiciels de haut niveau.
Le lancement du procédé permet aux équipes et à leurs gérants de :

- établir les objectifs ;
- définir les rôles des équipes ;
- évaluer les risques ;
- produire un plan d'équipe.

Après le lancement du procédé, TSP produit un "framework" de procédé défini pour gérer, tracer et rapporter les progrès de l'équipe.

En utilisant TSP, l'organisation peut construire des équipes auto dirigées qui planifient et tracent leurs travaux, établissent les objectifs, et possèdent leurs procédés et leurs plans.

[15] Extrait de [CM TSP]

TSP aide l'organisation à établir des pratiques matures et des disciplines produisant des logiciels sécurisés et fiables. TSP est également utilisé comme une base pour de nouveaux frameworks de mesures pour les développeurs et les acheteurs de logiciels. Ceci correspond au projet ISAM (Integrated Software Acquisition Metrics [CM ISAM]).

2.2.5. Personal Software Process (PSP)[16]

PSP permet aux ingénieurs de :

- gérer la qualité de leurs projets ;
- s'attribuer des responsabilités ;
- améliorer l'estimation et la planification ;
- réduire les défauts dans les produits.

Le coût du personnel constitue 70 pour cent du coût du développement logiciel. Pour cela, les compétences et les habitudes de travail des ingénieurs déterminent largement les résultats du procédé de développement logiciel. Basé sur les pratiques du CMM (présenté dans la suite de ce document), le PSP peut être utilisé par les ingénieurs comme un guide vers une approche disciplinée et structurée pour le développement logiciel. PSP est préalable à une planification d'organisation pour introduire TSP.

PSP peut être appliqué à un grand nombre de parties du procédé de développement logiciel, incluant :

- le développement de petits programmes ;
- la définition de besoins ;
- la rédaction de documents ;
- les tests de systèmes ;
- la maintenance de systèmes ;
- l'amélioration de systèmes logiciels importants.

2.2.6. Synthèse sur les Procédés Semi-Formels

Les procédés semi-formels sont des procédés logiciels non exécutables, qui ont pour but de décrire la façon dont le logiciel va être développé. Ils recommandent fortement l'utilisation de procédé pour formaliser et modéliser le développement logiciel.

Les procédés semi-formels que nous avons présentés dans cette section, ont beaucoup apporté au domaine de la modélisation du développement logiciel. Ils ont eu beaucoup de succès en entreprise, surtout RUP[17], qui présente les meilleures pratiques de développement. Ceci indique le grand intérêt qu'ont ces meilleures pratiques chez les industriels.

[16] Extrait de [CM PSP]

[17] Plusieurs centaines de milliers d'exemplaires de livres sur RUP ont été achetés, ce qui est rare en ce qui concerne les livres d'informatique.

De notre point de vue, l'utilisation de procédés est le meilleur moyen pour réaliser un travail de manière bien organisée, que ce soit pour faire du développement logiciel, pour automatiser les tâches à exécuter dans un système logiciel, dans le domaine de la gestion, etc. en effet, le procédé offre de bons moyens pour modéliser et de décrire ce travail.

Cependant, nous considérons que l'utilisateur a besoin d'un support automatisé lorsqu'il utilise le procédé. C'est pour cette raison que nous avons développé un environnement, permettant de construire et exécuter des procédés afin d'offrir à l'utilisateur une assistance efficace.

2.3. Les Procédés Organisationnels

Dans cette classe de procédés logiciels, on trouve :

- CMM (Capability Maturity Model) ;
- CMMI (Capability Maturity Model Integration) ;
- SPICE (Software Process Improvement and Capability dEtermination).

2.3.1. CMM (Capability Maturity Model) [18]

Le modèle d'évolution des capacités logiciel CMM a été développé par le SEI (Software Engineering Institute) afin de guider les organisations dans l'identification des stratégies d'amélioration des procédés par l'évaluation de la maturité[19] de leur procédé actuel et l'identification des quelques aspects critiques à la qualité logiciel et d'amélioration des procédés.

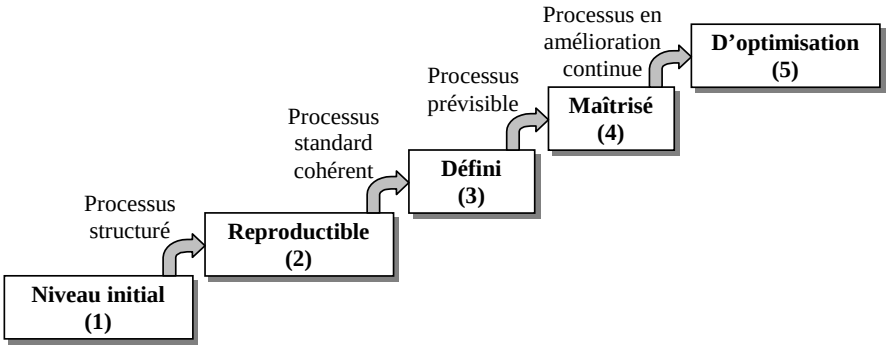


Figure 9 : Les cinq niveaux de maturité du procédé logiciel

CMM est composé de cinq niveaux de maturité. Un niveau de maturité est un palier d'évolution bien défini dans le cheminement vers un procédé logiciel mature. La structure à cinq niveaux du CMM (Figure 9) établit les priorités des actions d'amélioration en vue d'accroître la maturité du procédé logiciel. Les flèches dans la Figure 9 indiquent le type de

[18] Extrait de [PCCW 93] et de sa traduction [PCCW 93 fr]

[19] La maturité du procédé logiciel : désigne dans quelle mesure un procédé est explicitement défini, géré, mesuré, contrôlé et efficace.

capacité[20] en cours d'institutionnalisation dans l'organisation à chaque phase du cadre d'évolution.

1) Les cinq niveaux de maturité du procédé logiciel

La description des cinq niveaux ci-dessous souligne les changements apportés au procédé primaire à chaque niveau :

Niveau initial

Le procédé logiciel est caractérisé par la prédominance d'interventions ponctuelles, voir chaotiques. Très peu de procédés sont définis et la réussite dépend de l'effort individuel.

Niveau reproductible

Une gestion de projet élémentaire est définie pour assurer le suivi des coûts, des délais et de la fonctionnalité du produit. La discipline nécessaire au procédé est en place pour reproduire la réussite de projets d'un même domaine d'application.

Niveau défini

Le procédé logiciel des activités de gestion et d'ingénierie est documenté, normalisé et intégré dans le procédé logiciel standard de l'organisation. Tout nouveau projet de développement ou de maintenance logiciel fait intervenir une version adaptée et approuvée du procédé logiciel standard de l'organisation.

Niveau maîtrisé

Des mesures détaillées sont prises en ce qui concerne le déroulement du procédé logiciel et la qualité des produits de travail logiciel. Le procédé logiciel et le niveau de qualité des produits sont connus et contrôlés quantitativement.

Niveau d'optimisation

Une amélioration continue du procédé logiciel est mise en œuvre par une rétroaction quantitative émanant du procédé lui-même et par l'application d'idées et de technologies innovatrices.

2) Structure interne des niveaux de maturité

À l'exception du Niveau 1, chaque niveau de maturité se décompose en plusieurs ***secteurs clés***. Chacun de ces secteurs clés comprend cinq parties appelées ***caractéristiques communes***. Ces caractéristiques communes indiquent les ***pratiques clés*** qui, traitées collectivement, permettent d'atteindre les objectifs du secteur clé.

[20] La capacité du procédé logiciel : la gamme des résultats attendus pouvant être obtenus en suivant un procédé logiciel.

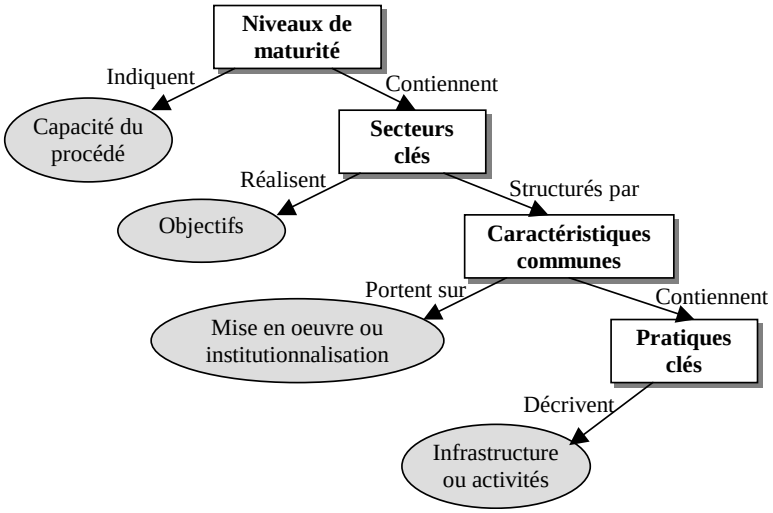


Figure 10 : Structure du CMM

La Table 1 montre les secteurs clés pour chaque niveau de maturité.

Niveau de Maturité	Secteurs clés
Reproductible (2)	Gestion des exigences Planification de projet logiciel Suivi et supervision de projet logiciel Gestion de la sous-traitance logiciel Assurance-qualité logiciel Gestion de configuration logiciel
Défini (3)	Focalisation organisationnelle sur les procédés Définition du procédé de l'organisation Programme de formation Gestion logiciel intégrée Ingénierie de produits logiciel Coordination intergroupes Revues par les pairs
Maîtrisé (4)	Gestion quantitative de procédé Gestion de la qualité logiciel
Optimisé (5)	Prévention des défauts Gestion des changements technologiques Gestion des changements du procédé

Table 1 : Secteurs clés pour chaque niveau de maturité

Chaque secteur clé est décrit en termes de pratiques clés contribuant à en satisfaire les objectifs. Les pratiques clés décrivent l'infrastructure et les activités qui contribuent le plus à la mise en œuvre et à l'institutionnalisation efficaces du secteur clé. Dans tout le modèle CMM, on compte plus de 300 pratiques clés.

Les pratiques clés de chaque secteur clé sont réparties parmi cinq caractéristiques communes. Ces caractéristiques communes sont des attributs indiquant si la mise en oeuvre et l'institutionnalisation d'un secteur clé sont ou non efficaces, reproductibles et durables. Les cinq caractéristiques communes intervenant dans le CMM sont les suivantes :

- Engagement de réalisation
- Capacité de réalisation
- Activités réalisées
- Mesures et analyse
- Vérification de mise en oeuvre

2.3.2. CMMI (Capability Maturity Model Integration)[21]

Récemment, le SEI (Software Engineering Institute) a amélioré son modèle d'évolution des capacités logiciel (CMM) vers le modèle intégré d'évolution des capacités logiciel (CMMI Capability Maturity Model Integration). Ce modèle est basé sur les meilleures pratiques du CMM. Il les étend, afin de compléter le programme d'amélioration de procédé. CMMI représente ainsi un niveau plus élevé du procédé de capacité et de maturité.

L'adoption de ce modèle par les industriels a été assez rapide. Des centaines d'organisations sont passées au CMMI dans le monde entier. Vers la fin de l'année 2002, le SEI a proposé des cours d'introduction au CMMI, auxquels 7000 personnes ont participé, ce qui montre le grand intérêt qu'a eu le CMMI auprès des industriels.

2.3.3. SPICE (Software Process Improvement and Capability dEtermination)[22]

Le projet SPICE est une initiative de l'ISO (l'Organisation Internationale de Normalisation [ISO]) et de l'IEC (la Commission Electrotechnique Internationale [IEC]). Cette initiative a pour but le développement d'un Standard International pour l'estimation et l'évaluation des procédés logiciels.

Bien que SPICE s'inspire largement de CMM, il existe plusieurs différences entre eux. La différence la plus importante est le fait que le modèle CMM s'applique à une organisation dans son ensemble, alors que SPICE décrit la maturité de chaque processus individuellement.

2.3.4. Synthèse sur le Procédé Organisationnel

CMM et CMMI considèrent que le développement logiciel ne peut être mature et efficace que s'il est établi sur une infrastructure de procédé fondée sur des pratiques efficaces d'ingénierie logiciel et de gestion. Ils définissent alors les niveaux de maturité, et décrivent les éléments clés d'un procédé logiciel efficace.

[21] Extrait de [CMMI], [CMMI 03]

[22] Extrait de [CM SPICE], [EDMD 97], [ELCA 05]

CMM et CMMI ont eu un énorme succès auprès des industriels, ils ont réussi à faire entrer le procédé dans l'industrie, et à ancrer dans les esprits l'importance de se baser sur un procédé lors du développement logiciel.

SPICE a eu moins de succès que CMM et CMMI. Cependant, de nombreux efforts internationaux sont mis en place pour améliorer le projet SPICE et le diffuser, à travers des formations et des tutoriaux. Le SEI (Software Engineering Institute – développeur du CMM et du CMMI), participe fortement à ces efforts d'amélioration et de diffusion de SPICE. Il joue également un rôle d'évaluateur des méthodes proposées dans le projet SPICE.

2.4. Les Procédés de Gestion de Configuration (SCM Software Configuration Management)[23]

La gestion de configuration est le contrôle de l'évolution des systèmes complexes. Un système complexe est un système ayant :

- un grand nombre de composants, et de versions de composants (des milliers) ;
- un grand nombre de personnes impliquées (des dizaines ou des centaines) ;
- des contraintes strictes de temps (cycles de quelques mois) ;
- une longue durée de vie (jusqu'à 10 ou 20 ans).

L'objectif de la gestion de configuration est de contrôler l'évolution du système, et d'aider à satisfaire les contraintes de délais et de qualité.

La gestion de configuration est apparue au début des années 80. A cette époque, elle s'intéressait à la programmation globale (*programming in the large*). Pendant les années 90, la gestion de configuration s'est concentrée sur la programmation coopérative (*programming in the many*), et à la fin des années 90, elle s'est concentrée sur la programmation distribuée (*programming in the wide*).

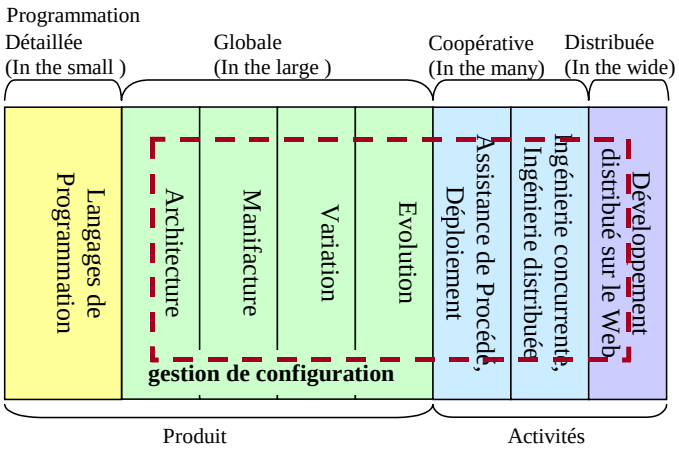


Figure 11 : Génie Logiciel, Programmation, et Gestion de Configuration

[23] Extrait de [ELC 02], et de [Est 00]

2.4.1. Apports de la gestion de configuration

Nous pouvons distinguer trois sujets principaux auxquels s'intéresse la gestion de configuration aujourd'hui :

1) La gestion de produits logiciels, à travers le versionnement, et la modélisation de produits ;
2) L'assistance aux utilisateurs et aux outils, à travers la gestion de l'espace de travail[24], et les outils de compilation du logiciel ;
3) Le contrôle des changements.

Nous nous intéressons plus particulièrement au troisième point.

Le Contrôle de Changements (Change Control)

Puisque la gestion de configuration se charge de contrôler l'évolution du logiciel, elle est fortement liée au procédé contrôlant le cycle de vie de ce logiciel au cours de sa création et de son évolution. Le contrôle du cycle de vie du logiciel via un procédé permet de contrôler la qualité, le coût et les délais de construction du logiciel. Ce contrôle consiste à décrire formellement l'évolution du logiciel dans un modèle de procédé, et à exécuter ce procédé afin de fournir une assistance automatisée aux développeurs du logiciel dans leurs tâches quotidiennes.

Les procédés de gestion de configuration contrôlent les changements du produit logiciel, ces changements sont soit des demandes de modifications, soit des défaillances à réparer. Les premiers systèmes de contrôle de changements automatisés imposaient des procédés spécifiques de contrôle de changement. Par la suite, ces systèmes ont permis aux développeurs de créer leurs propres procédés, utilisant des formalismes basés sur un diagramme de transition d'état. Un diagramme de transition d'état décrit pour chaque type d'objets : la succession d'états permise pour cet objet et les opérations valides dans chaque état, et précise quelle action produit une transition d'un état à l'autre.

Beaucoup de formalismes de modélisation de procédé ont été proposés, afin de permettre aux utilisateurs de créer leurs propres procédés, pour contrôler le cycle de vie de leurs logiciels. Cependant, beaucoup d'utilisateurs ont préféré acheter des modèles de procédés déjà définis, contenant les "meilleures pratiques", plutôt que de modéliser leur propre procédé.

Bien que les modèles de procédé dans la gestion de configuration ne concernent que le contrôle des changements du produit logiciel, ils ont aussi été utilisés pour gérer les espaces de travail, afin de gérer les activités d'ingénierie concurrente.

[24] L'espace de travail est une partie du système de fichiers du programmeur où se trouvent les fichiers et les objets du produit logiciel dont il a besoin afin d'accomplir ses tâches.

2.4.2. Synthèse sur les Procédés de Gestion de Configuration

La préférence qu'ont eu les utilisateurs à acheter des modèles de procédés déjà définis, plutôt que de modéliser leurs propres procédés, peut être interprétée de plusieurs manières :

- Cette technologie est venue en avance par rapport à la maturité des utilisateurs ;
- Les concepts et les interfaces proposés pour modéliser les procédés étaient trop complexes ;
- Les utilisateurs ont eu des difficultés à identifier les bons procédés.

Les utilisateurs ont donc trouvé la solution en achetant des modèles de procédés définis par des experts. Ces modèles contenant les meilleures pratiques ont eu beaucoup de succès par le fait qu'ils correspondaient à la réalité et parce qu'il n'est pas simple de définir son propre modèle de procédé.

Pour élargir l'utilisation à des modèles de procédé fait par les développeurs eux même, un travail de simplification des concepts proposés pour modéliser les procédés doit être fait.

2.5. Les Procédés Logiciels récents : SPEM

Nous présentons, pour les procédés logiciels récents, SPEM (Software Process Engineering Metamodel) [SPEM 05].

SPEM est une proposition de l'OMG qui définit un formalisme dédié à la description du processus de développement logiciel. Les outils basés sur SPEM sont des outils de rédaction et de personnalisation de processus. La planification et l'exécution d'un projet décrit avec SPEM ne rentrent pas dans le domaine de ce modèle.

SPEM est à la fois un méta modèle MOF et un profil UML, des correspondances ont été établies entre ces deux formalismes.

2.5.1. Les éléments de structure du procédé dans SPEM

SPEM contient plusieurs éléments de structure du procédé, permettant de construire une description de procédé. Ces éléments sont assez similaires aux éléments du méta-modèle de Procédé Logiciel présentes dans la section 2.1.1 de ce document (Figure 5) :
Un procédé *(Process)* peut être considéré comme une suite d'opérations *(WorkDefinition)*, réalisées par des rôles *(ProcessRole)*, qui vont utiliser et consommer des produits *(WorkProduct)*, et des informations *(InformationElement)*. Ces produits et informations constituent les paramètres de l'activité. L'activité est contenue par le rôle qui la réalise.

2.5.2. Les éléments de définition du procédé dans SPEM

SPEM contient des éléments de définition permettant de définir par quelle manière va s'exécuter le procédé. Ces éléments sont les suivants :

L'*Etape (Step).* Une étape est une définition de travail[25] atomique, à grain fin, utilisée pour décomposer des Activités.

L'*Activité (Activity).* Définition d'un travail décrivant ce qui est produit dans la cadre d'un Rôle . Les activités sont l'élément principal d'un travail.

L'*Itération (Iteration).* Une Itération est une définition de travail, à gros grain, qui représente un ensemble d'Activités, visant au développement d'une portion du système, et qui s'achève par la fourniture (interne ou externe) d'un produit logiciel.

La *Phase (Phase).* Définition à haut niveau d'un travail, qui se termine par un jalon.

Le *Cycle de vie (Life Cycle).* Un cycle de vie pour un processus est défini comme une séquence de phases pour accomplir un but précis. Le cycle de vie définit le processus qui sera appliqué sur un projet donné.

La Figure 12 présente une vue partielle du Meta-Modèle de SPEM.

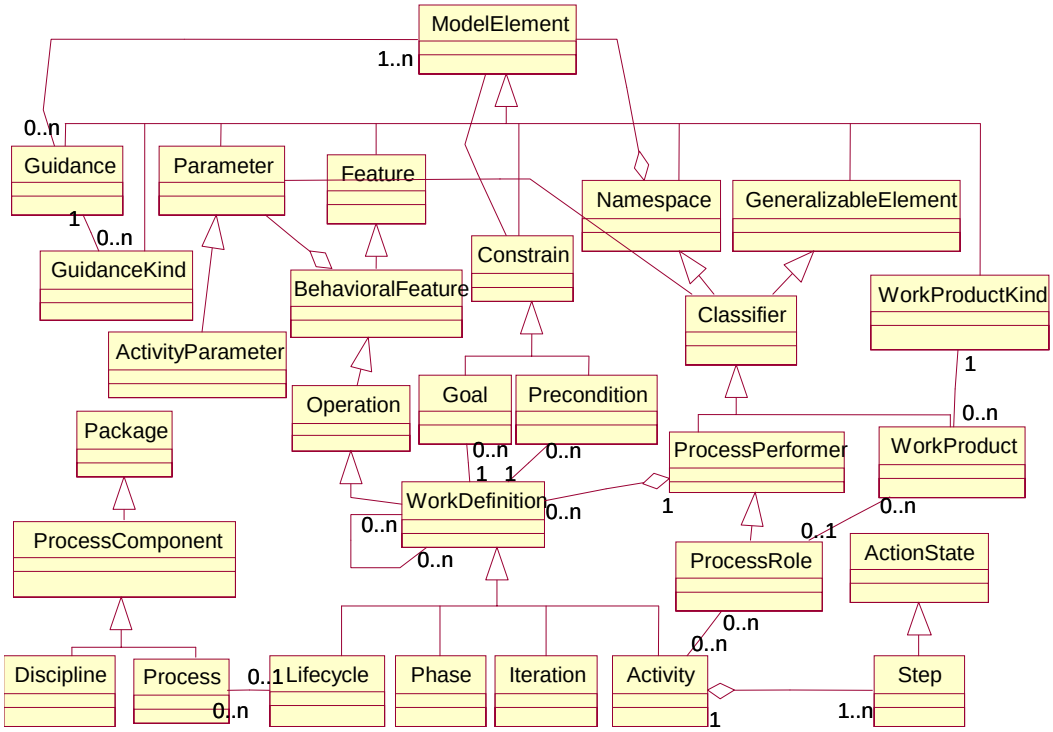


Figure 12 : vue partielle du Meta-Modèle de SPEM

La Table 2 présente les correspondances entre les éléments de structure et de définition du procédé dans le méta-modèle SPEM, les éléments du méta-modèle de Procédé Logiciel, et les éléments du modèle RUP.

SPEM	ProcessRole	Activity	WorkProduct Information-	Discipline	Lifecycle	Phase	Iteration

[25] Une définition de travail est un élément qui décrit l'exécution, les opérations réalisées, et les transformations opérées sur les produits de travail dans le cadre d'un Rôle.

		Step	Element				
Procédé Logiciel	Role	Activity	Product				Iteration
Rational Unified Process	Role	Activity Step	Artifact	Discipline	Process	Phase	Iteration

Table 2 : correspondances entre les éléments de structure de SPEM, du Procédé Logiciel, et de RUP

2.5.3. Synthèse

SPEM est un formalisme dédié à la description du processus de développement logiciel. Il ne s'adresse pas à la planification ni à l'exécution d'un projet.

Percevant la surabondance des méta-modèles de procédé, Jean Bézivin, dans son article [BB 04], proposes d'utiliser les transformations techniques du MDE [Bez 02] afin de les rendre "compatible". Il propose de transformer un modèle SPEM, définis comme un profil UML, vers un modèle MS-Project, ce qui parait raisonnable puisque SPEM a été conçu avec les modèles MS-Project en tête. Mais ce travail n'est qu'une proposition théorique.

3. CSCW : Computer Supported Co-operative Work

Claude Godart à défini le CSCW comme étant la : "Collaboration à l'aide de l'ordinateur en vue d'augmenter la productivité et/ou la fonctionnalité des processus personnes à personnes" [God 02].

Le CSCW, en français le TCAO *(Travail Coopératif Assisté par Ordinateur)*, est un domaine multidisciplinaire incluant la sociologie, la psychologie, la linguistique et les sciences d'ordinateur. Son but est de comprendre et d'améliorer la productivité des personnes ou d'un groupe de personnes. Les applications CSCW (brièvement : les applications collaboratives) sont des systèmes logiciels permettant à de multiple utilisateurs de travailler ensemble. De part sa nature multi-utilisateur/multi-site, les applications CSCW s'appuie sur des implémentations et des architectures logicielles distribuées [CVK 99].
Beaucoup de travaux ont été réalisés, pour assister les développeurs des applications collaboratives. Ces travaux ont regroupé les fonctionnalités clés et les services communs dans une plate-forme de support, nommée "groupware", afin de réduire les efforts d'implémentation.

3.1. La terminologie du travail coopératif [26]

L'activité collaborative basée sur ordinateur se nomme la *session*. L'organisation d'une session requiert un mécanisme de programmation d'événements, ce que l'on appèle la *gestion de session*. L'*initiateur* démarre la session en l'enregistrant dans le système, et les autres *participants* se connectent simplement à la session, afin d'y prendre part. Le mécanisme

[26] Extraits de [CVK 99]

supportant la session pendant son exécution (la gestion de l'espace de travail partagé) s'appelle le *support actif de la session*. L'espace de travail partagé est l'environnement dans lequel se passe la collaboration.

Ellis et al. [EGR 91] ont classifiés les systèmes CSCW en quatre catégories, suivant la localisation des personnes participantes à l'interaction (même localisation ou différentes localisations) et les caractéristiques de l'interaction (les personnes interagissent en même temps ou à des moment différents). La Table 3 présente des exemples de ces quatre catégories.

	Même moment	**Différents moments**
Même localisation	Aide à la réunion : • Rétro-projections d'écrans • Tableaux blancs	Aide mémoire : • Base de documents • Data warehouse (entrepôt de données)
Différentes localisations	Réunion virtuelle : • Vidéo/visio conférence • Contrôle d'applications à distance • Editeurs synchrones	• Courrier électronique • Intranet • Workflow • Editeurs asynchrones

Table 3 : exemples des quatre catégories des systèmes CSCW [God 02]

3.2. La classification des systèmes CSCW [27]

D. Port et G. Kaiser [PK 98] ont classifié les systèmes de collaboration CSCW en quatre catégories, orthogonales aux catégories de Ellis et al. que nous avons présentés dans la section précédente. Ces quatre catégories sont :

1. L'ingénierie d'informations collaboratives (collaborative information engineering)

Ce domaine s'occupe de la capture dynamique d'information, de la distillation (raffinement d'informations à travers le filtrage collaboratif), et de l'utilisation des informations venant de divers groupes distribués.

2. Les procédés et les workflows collaboratifs

Ceci est un domaine vaste, couvrant les problèmes relatifs à la gestion des activités collaboratives, à travers une spécification de procédé offrant un support d'assistance. Ceci inclus les problèmes classiques de concurrence, de gestion de configuration, de versionnement, de gestion de transaction, de distribution, et de livraison/transport d'objets.

3. Le support de collaboration

[27] Extraits de [PK 98]

Ce domaine s'occupe des fondements des procédés collaboratifs distribués. Il fournit les moyens pour capturer et diffuser des informations basiques de procédés collaboratifs, comme par exemple l'objectif et le statut.

4. L'intégration des systèmes collaboratifs

Le but des environnements collaboratifs intégrés est de produire un framework général, permettant aux systèmes de collaboration et aux outils les soutenant d'interagir et de partager des informations sans que ces systèmes se connaissent.

3.3. Synthèse sur CSCW

Le CSCW est un domaine qui s'intéresse à la gestion du travail coopératif, à l'aide d'outils informatique. Le CSCW s'intéresse surtout à la rédaction de document de manière collaborative. Le but des outils du CSCW est alors de donner l'illusion que tout le monde écrit dans le même document. Par conséquent, beaucoup de personnes, en même temps, utilisent la même donnée, ce qui fait la différence entre le CSCW et les autres domaines.

Les procédés ont intégré le domaine du CSCW, afin d'être utilisés pour la gestion de ces activités collaboratives. Ainsi, l'utilisation des procédés a pris une forme différente de celle qu'elle avait dans le domaine des Procédés Logiciels :

- Dans le domaine des Procédés Logiciels, le procédé est utilisé pour gérer et assister le développement et l'évolution du logiciel. Le produit résultant de l'utilisation du procédé est le système logiciel ;
- Dans le CSCW, le procédé est une partie intégrante du système logiciel, et participe à son exécution. Pendant l'exécution du logiciel de CSCW, le procédé gère la collaboration en coordonnant les activités collaboratives.

Les applications de CSCW, en elles-mêmes, ne sont pas très représentatives de ce que nous cherchons à faire. Cependant, nous nous intéressons aux procédés intégrés aux systèmes logiciels, et, comme nous l'avons vu dans cette section, c'est dans le domaine du CSCW que les procédés ont commencé à constituer une partie intégrante du système logiciel ; leur exécution permettant de formaliser la coordination des tâches composant le système logiciel.

Cependant, les formalismes décrivant les procédés de collaborations manquent de maturité, notamment en ce qui concerne l'évolution du procédé, qui est un point très important dans la vie d'un système logiciel. Le formalisme de description de procédé doit permettre l'introduction de nouveaux concepts, pour permettre l'évolution et l'amélioration du procédé. Ce formalisme doit également être simple, compréhensible, afin de permettre à ses utilisateurs de manipuler ses concepts sans difficulté.

Ces deux caractéristiques sont assez contradictoires, car en facilitant l'introduction de nouveau concepts dans le formalisme afin de le faire évoluer, il devient rapidement très

complexe et difficile à gérer. Ceci est un problème que nous cherchons à résoudre dans cette thèse.

4. Les Workflows

Le nom "Workflow" est utilisé dans l'industrie, pour designer les procédés industriels. Les vendeurs de logiciels produisant des produits de workflow, définissent les workflows de plusieurs manières [GHS 95] :

- "Le workflow est le mécanisme à travers lequel on peut implémenter des pratiques de re-ingénierie métier" (PeopleSoft Inc. [Frye 94]) ;
- Le workflow est "le travail reformulé en forme d'une série de transactions basée sur les personnes", "une série de workflows forme un procédé métier" (Action Technologies Inc. [Frye 94]) ;
- "le workflow est le procédé par lequel les tâches individuelles sont assemblées pour compléter une transaction – un procédé métier clairement définis – à l'intérieur d'une entreprise" (Recognition Internal Inc. [AT 93]) ;
- "Le workflow va au-delà de l'acheminement [c.-à-d. transporter les informations entre les utilisateurs ou les systèmes] en intégrant les informations de diverses sources" (les laboratoires Wang [Frye 94]).

Les procédés ont existé dès le début de l'industrialisation. Ils sont le résultat des recherches pour augmenter l'efficacité en ce concentrant sur les aspects répétitifs du travail. Grâce aux procédés, le travail a été séparé en tâches, rôles, règles et procédures bien définies, ce qui contrôlait la plupart du travail dans les industries et les bureaux. Initialement, les procédés étaient entièrement effectués par des humains manipulant des objets physiques. Avec l'introduction des technologies de l'information, les procédés dans les lieux de travail ont été partiellement ou totalement automatisé, c.-à-d. des programmes informatiques exécutaient les tâches et faisaient respecter les règles définies auparavant par les humains. Au début, les ambitions étaient très réduites et les procédés étaient simples et déterministes, ne faisant que de l'automatisation de bureautique (office automation). Ces ambitions se sont progressivement élargies et les procédés dans l'industrie ont graduellement évolués.

4.1. Les catégories des Workflows [28]

La presse industrielle caractérise les workflows dans trois catégories [GHS 95] : le workflow improvisé (ad hoc), le workflow administratif, et le workflow de production (cette caractérisation a été faite en premier par McCready [Mc 92]). La Figure 13 présente les trois catégories de workflows d'après la presse industrielle.

[28] Extrait de [GHS 95]

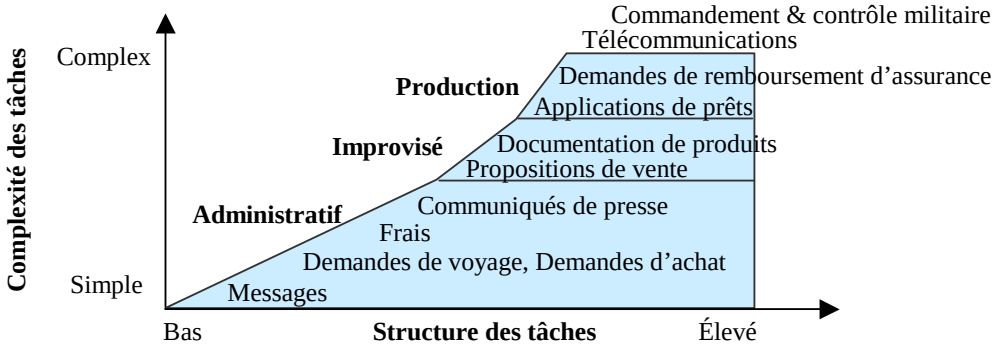


Figure 13 : les trois catégories de workflows d'après la presse industrielle [GHS 95]

4.1.1. Les Workflows improvisés (Ad hoc)

Les workflows improvisés exécutent des procédés de bureau, comme les documentations de produits ou les propositions de vente. D'habitude, les tâches du workflow improvisé impliquent la coordination, la collaboration, ou la co-décision des humains. Les décisions d'ordonnancement et de coordination des tâches sont faites pendant l'exécution du workflow.

Les WFMSs[29] soutenant les workflows improvisés doivent produire des fonctionnalités pour faciliter la coordination, la collaboration, et la co-décision des humains. Les fonctionnalités pour contrôler l'ordre des tâches ne sont généralement pas fournies dans ce genre de WFMSs. Les utilisateurs d'un workflow improvisé ont besoin d'accéder au WFMS pour déterminer si le travail a été achevé.

Les WFMSs improvisés ne sont pas des systèmes critiques[30], c-à-d., une panne répétée de ce genre de workflows n'interfère pas vraiment avec le procédé métier en entier. Les WFMSs improvisés utilisent d'habitude une base de données (propriétaire) pour stocker les informations partagées (ex. les documents comme les formulaires de relecture pour les conférences ou les articles). Les WFMSs assistant les workflows improvisés sont aussi appelés *groupware* (nous avons parlé des groupwares dans la section 3 concernant le domaine du CSCW).

La Figure 14 illustre un workflow improvisé simplifié de relecture d'articles de conférence. Le procédé de relecture se déroule de la manière suivante : les relecteurs sont sélectionnés,

[29] WFMS: système de gestion de workflow (Workflow Management System), voir 4.2

[30] Un système est dit critique lorsque : (1) la vie de personnes est tributaire de son fonctionnement : logiciels embarqués pour le transport (avion, train...), contrôleurs de systèmes (centrales nucléaires, matériels médicaux...) ; (2) le coût économique d'un dysfonctionnement est catastrophique : logiciels mis dans le silicium et produit en grande quantité (électroménager, téléphonie...) [Gri ISC]. Anyanwu et al. définissent un système critique dans leur article [ASCMK 03] comme étant un système exigeant de la robustesse, un bon support de traitement d'exceptions, une grande capacité de passage à l'échelle, de la gestion de la qualité de service, et la possibilité de faire des modifications dynamiques. Ils présentent dans cet article un système critique utilisé dans le domaine médical.

des articles à relire leurs sont distribués, une fois les relectures achevées, ils collaborent pour produire un assemblage des relectures, ce document assemblé est finalement envoyé aux auteurs. Ceci est un workflow improvisé car il implique :

1) la négociation pour sélectionner les relecteurs ; et
2) la collaboration entre les relecteurs pour produire le document assemblé de relectures.

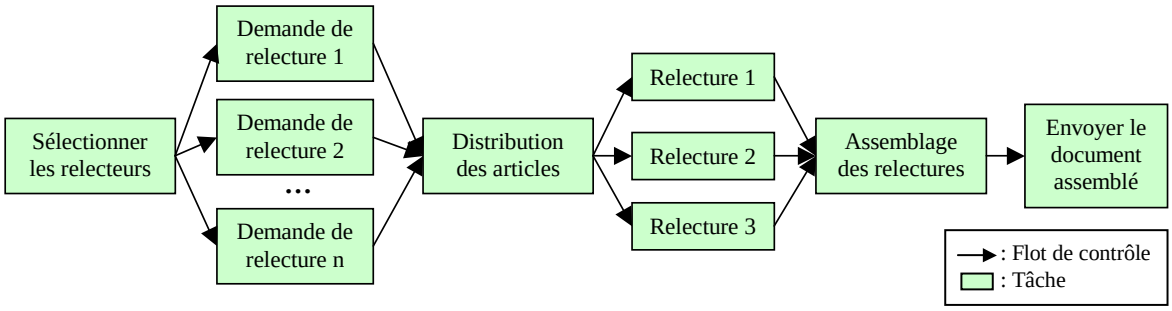


Figure 14 : un workflow improvisé de relecture d'articles de conférence [GHS 95]

4.1.2. Les Workflows administratifs

Les workflows administratifs impliquent des procédés répétitifs et prévisibles, avec des règles de coordination de tâches simples, comme l'acheminement d'un état de frais ou une demande de voyage à travers un procédé d'autorisation.

Le contrôle et la coordination des tâches dans les workflows administratifs peut être automatisé. Un WFMS supportant un workflow administratif, comprend de simples informations pour l'acheminement, et des fonctions d'acceptation du document, comme celles que l'on trouve dans les planifications de voyages et les demandes d'achat.

Les workflows administratifs ne contiennent pas de procédé d'informations complexes, et n'ont pas besoin d'accéder à de multiples systèmes d'information utilisés pour prendre en charge la production et/ou les services clientèle. Le WFMS administratif n'est généralement pas un système critique.

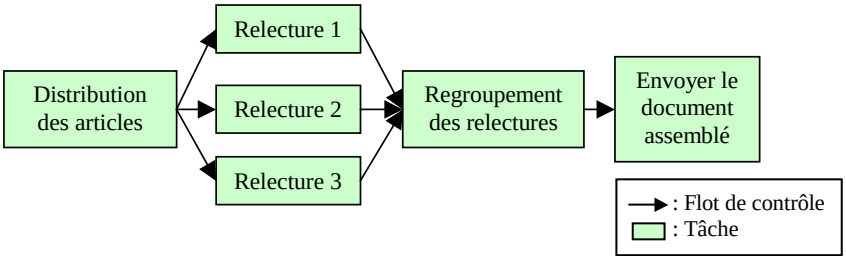


Figure 15 : un workflow administratif de relecture d'articles de conférence [GHS 95]

On reprend le procédé de relecture d'articles de conférence, mais cette fois-ci, on suppose que les relecteurs sont connus à l'avance (ex. les mêmes relecteurs relisent tous les articles). On suppose également que les relecteurs ne collaborent pas pour produire un assemblage des relectures. A la place, ils produisent des relectures individuelles examinées par l'éditeur (ex. le président du programme) qui prend la décision finale. Avec ces suppositions-là, le

workflow de relecture d'articles devient un workflow administratif, comme le représente la Figure 15.
Dans un workflow administratif, les utilisateurs sont sérieusement incités à accomplir leurs tâches. Alors que les relecteurs utilisant un workflow improvisé ont besoin d'accéder au WFMS pour déterminer si le travail a été accompli, les relecteurs utilisant un WFMS administratif peuvent recevoir un courrier électronique contenant les instructions de relecture, accompagnés de l'article à relire et le formulaire des commentaires de relecture. Quand le formulaire est rempli, il est acheminé automatiquement vers le président du comité du programme qui est averti lorsque toutes les relectures sont accomplies.

4.1.3. Les Workflows de production

Les workflows de production impliquent les procédés métiers répétitifs et prévisibles, comme les applications de prêts ou de demandes de remboursement d'assurance. Contrairement au workflow administratif, le workflow de production contient d'habitude un procédé d'information complexe impliquant l'accès à de multiples systèmes d'information. L'ordonnancement et la coordination des tâches dans un tel workflow peut être automatisé.

Toutefois, l'automatisation des workflows de production est compliquée à cause :

1) de la complexité du procédé d'information ; et
2) des accès aux multiples systèmes d'information afin de récupérer des données pour prendre des décisions.

Les WFMSs supportant les workflows de production doivent faciliter la définition des dépendances des tâches, et l'exécution des tâches de contrôle avec peu ou pas d'intervention humaine. Les WFMSs de production sont souvent des système critiques et doivent gérer l'intégration et l'interopérabilité des systèmes d'information HAD (Hétérogènes, Autonomes, et/ou Distribués).

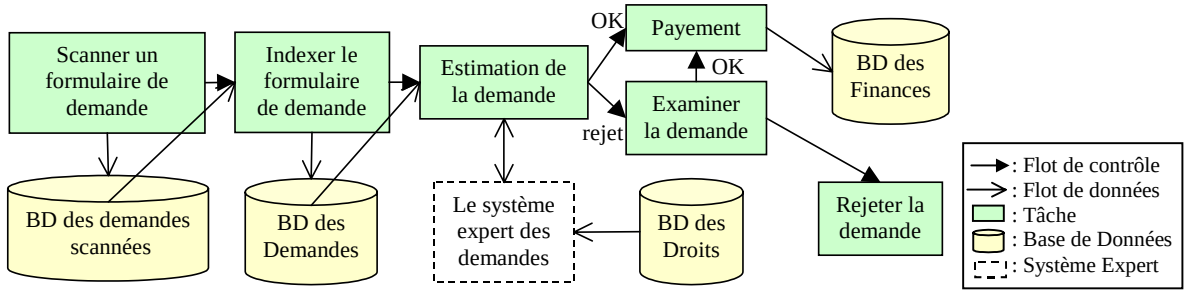


Figure 16 : un workflow de "procédé de demandes sanitaires" [GHS 95]

La Figure 16 représente un workflow de "procédé de demandes sanitaires" simplifié. Dans ce workflow, un formulaire de demande est d'abord scanné manuellement et stocké dans une base de données objet. Ensuite, la demande est indexée manuellement dans une base de données relationnelle. Puis, cette information est analysée par une tâche automatisée : "Estimation de la demande". Cette tâche est exécutée par un système expert utilisant une base de données de droits pour déterminer si le paiement doit avoir lieu. Si la

demande est rejetée, un représentant discute avec le client de sa demande. A l'issue de cette discussion, le représentant soit accepte de payer une partie, soit de rejeter la demande. Si le paiement est effectué, la tâche "Payement" accède à la base de données des finances et enregistre le paiement.

Les différences essentielles entre ce workflow de production et les deux autres types de workflows (le workflow improvisé et le workflow administratif), sont :

1) l'interaction des systèmes d'information avec le procédé métier ; et
2) l'utilisation de tâches automatisées (non humaines) pour l'exécution.

4.2. Les Systèmes de Gestion de Workflow (WFMSs) [31]

Les systèmes de gestion de workflow (WFMSs Workflow Management Systems) permettent de spécifier, exécuter, faire des comptes-rendu, et contrôler dynamiquement les workflows. Nous présentons dans cette section les caractéristiques et les capacités supportées actuellement par les Systèmes de Gestion de Workflow, ainsi que leurs limitations.

4.2.1. Les caractéristiques supportées par les WFMSs

Les caractéristiques et les fonctionnalités supportées par les WFMSs commerciaux se résument dans les points suivants :

1) Le modèle de workflow (souvant basé activités) ;
2) Le langage de spécification (langage graphique de spécification, souvent basés sur des règles ou sur des contraintes) ;
3) Les outils pour tester/analyser et superviser (monitoring) ;
4) Les architectures du système et l'interopérabilité (basés soit sur le courrier électronique, soit sur le stockage dans une base de données partagée, soit comme beaucoup le font en combinant les deux) ;
5) Le support d'implémentation (la modification dynamique, la signalisation des événements et la notification, et l'administration d'utilisateurs).

4.2.2. Les limitations des WFMSs commerciaux

Malgré leurs bonnes propriétés, les WFMSs contiennent un nombre important de limitations. Ces limitations incluent :

- le manque d'interopérabilité entre les WFMSs (à cause du manque de standards de WFMSs) ;
- le manque de prise en charge de l'interopérabilité entre les systèmes HAD[32] ou entre un WFMS et des systèmes HAD ;
- une performance insuffisante pour certains procédé métier, qui peuvent nécessiter la manipulation d'un grand nombre de workflows ;

[31] Extrait de [GHS 95]

[32] HAD : Hétérogènes, Autonomes, et/ou Distribués

- le manque de prise en compte de la cohérence et de la fiabilité concernant la gestion d'accès concurrentiels et de reprise sur pannes (important à cause du partage des ressources) ;
- la faiblesse des outils de prise en charge de l'analyse, du test, et du débogage de workflows.

4.3. Synthèse sur les Workflows

Au début de l'utilisation des workflows dans les entreprises, les ambitions étaient très réduites. Les workflows n'étaient pas automatisés mais contrôlés par les humains. Ces workflows ont été appelés "les workflows improvisés (Ad hoc)". Les workflows improvisés ont ensuite évolués vers des workflows automatisés, simples et déterministes, qui ont porté le nom de "workflows administratifs". Avec le temps, ces workflows sont devenus plus complexes. Ils ont commencé à utiliser des systèmes d'information pour automatiser la prise de décisions, et ils sont devenus de plus en plus critiques. Ceci a donné naissance à un troisième type de workflows : les workflows de production. A ce moment-là, lorsque les workflows sont devenus complexes, les industriels ont commencé à s'intéresser au travail qui a été fait dans le domaine des procédés logiciels.

La Figure 17 illustre la croissance des travaux autour des procédés logicels et des workflows. Dans cette figure, nous percevons que les travaux autour des procédés logicels ont démaré au debut des années 90, et ont acquis leur apogé au milieu des années 1990. Le nombre de traveaux autour des procédés logicels a ensuite progréssivement diminué. Les travaux autour des workflows, par contre, n'ont commencé que vers le milieu des années 90, et ils ont continué, par la suite, à croître et à se developper.

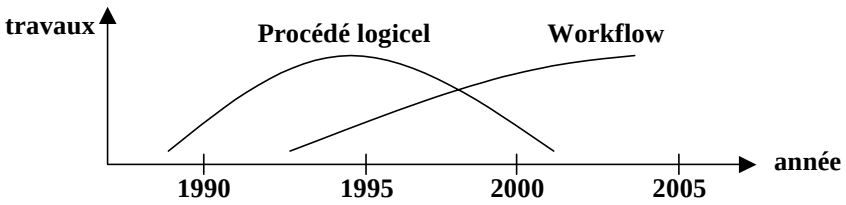


Figure 17 : croissance des travaux autour des procédés logicels et des workflows

Une organisation internationale nommée "la coalition de gestion de workflow" (WfMC Workflow Management Coalition) [WfMC], a été créée en août 1993, afin de regrouper les intéressés par les workflows, que ce soit des utilisateurs, des vendeurs, des analystes ou des universitaires. L'objectif principal de cette organisation a été de promouvoir et de développer l'utilisation des workflows.

Dans notre équipe, nous avons construit un environnement de gestion de procédé basé activité, construit à la base pour gérer des procédés logiciels exécutables. Notre environnement a ensuite évolué, pour gérer la construction et l'exécution de procédés

génériques, utilisés pour la coordination de tâches, de manière similaire aux WFMSs. Ces procédés génériques couvrent plusieurs domaines d'utilisation.

5. La Gestion de Projet (Project Management)

La Gestion de Projet est l'application de connaissances, compétences, outils et techniques aux activités d'un projet, afin d'atteindre les objectifs attendus du projet, et réaliser les exigences requises par les participants à ce projet [PMI 96].

La gestion de projet est une démarche visant à structurer, assurer et optimiser le bon déroulement d'un projet, l'objectif étant d'obtenir un résultat de qualité pour le moindre coût et dans le meilleur délai possible [Wiki GP].

Le travail du chef de projet logiciel varie suivant le produit logiciel développé. Néanmoins, la plupart des chefs de projet logiciel se chargent des activités de gestion suivantes [Som 04] :

- Ecriture de la proposition ;
- Planification et programmation du projet ;
- Evaluation des coûts du projet ;
- Contrôle et révision du projet ;
- Sélection et évaluation du personnel ;
- Ecriture du rapport et des présentations.

L'utilisation des procédés dans le domaine de la gestion de projet intervient surtout dans la planification et le développement du projet. Nous présentons ci-dessous cette activité.

5.1. La Planification et le Développement du projet

Dans la planification du projet, les activités du projet sont identifiées, ainsi que les jalons (milestones) et les délivrables produits par le projet. Un plan est établi afin de guider le développement vers les objectifs du projet.

Le plan du projet

Le plan du projet organise les ressources disponibles. Il découpe le travail à réaliser, et fournit un programme (un procédé de développement) pour accomplir ce travail.

Le développement du projet

Le développement du projet est l'une des activités les plus difficiles du gestionnaire de projet. Dans cette activité, le chef du projet :

- découpe le projet en tâches ;
- estime le temps et les ressources nécessaires à chaque tâche ;
- organise les tâches de façon concurrente afin d'optimiser le travail ; et
- minimise les dépendances entre les tâches afin d'éviter les éventuels retard provoqués par l'attente d'une tâche la terminaison d'une autre.

Les jalons

Pour gérer un projet correctement, il est indispensable de produire des rapports et des documents décrivant l'état du logiciel en cours de développement. Pour cela, il est important de définir un certain nombre de jalons, qui représentent les points finaux des activités du procédé. A chaque jalon, un rapport formel doit être produit.

Les phases et le cycle de vie du projet

Le projet est divisé en plusieurs phases, chaque phase du projet s'achève par la production d'un délivrable (les délivrables sont des résultats du projet, délivrés au client). L'ensemble des phases d'un projet est nommé : "le cycle de vie du projet".

5.2. Les procédés de gestion de projet

Les projets sont composés de procédés, exécutés par des personnes. Nous distinguons deux catégories de procédés, se chevauchant et interagissant tout au long du projet. Ces deux catégories sont :

- **Les procédés de gestion de projet** qui décrivent et organisent le travail dans le projet. Dans cette section, nous allons présenter cette catégorie de procédés ;
- **Les procédés orientés produit** qui spécifient et créent les produits du projet. Les procédés orientés produit sont typiquement définis par le cycle de vie du projet (voir **Les phases et le cycle de vie du projet** dans la section 5.1), et varient par domaine d'application.

Les procédés de gestion de projet peuvent être organisés en cinq groupes, chacun de ces groupes contient un procédé ou plus : les procédés d'initialisation ; les procédés de planification ; les procédés d'exécution ; les procédés de contrôle ; les procédés de fermeture. Ces groupes de procédés se chevauchent, et varient dans la phase qui les contient (Figure 18).

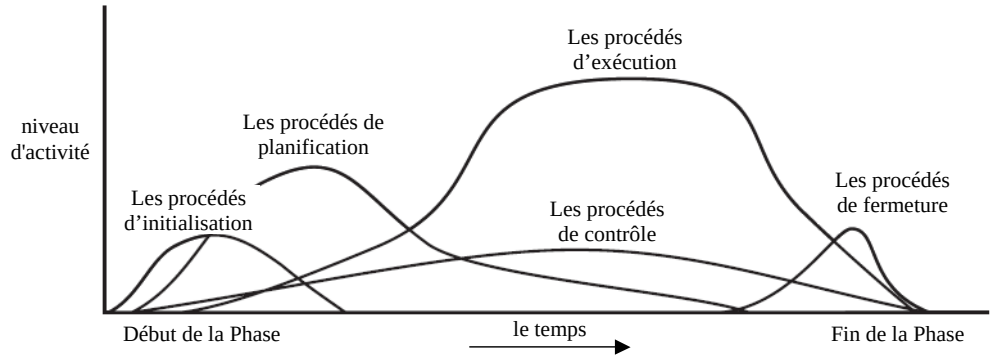


Figure 18 : Le chevauchement des groupes de procédés dans une phase

5.3. Exemple d'un environnement de gestion de projet : MS Project [33]

MS Project (Microsoft Office Project) est un produit de Microsoft, destiné à gérer les projets d'entreprises. Il offre une vision transversale sur l'ensemble des projets à gérer, et facilite la collaboration entre les différentes équipes des projets. Ceci afin d'aligner les ressources et les projets sur les objectifs de l'entreprise, d'identifier plus rapidement les problèmes, et de prendre des actions correctives.

MS Project contient plusieurs points intéressants pour la gestion de projets :

1. L'accès flexible aux informations et la centralisation de la gestion de documents, permettent aux équipes de collaborer et de suivre l'évolution du projet ;
2. Les utilisateurs ont la possibilité d'implémenter des standards et des modèles tirés de leur propre expérience ;
3. La gestion centralisée de ressources, les outils avancés d'optimisation de ressources, et les fonctions de planification de ressources, permettent d'obtenir des informations précises sur les compétences et l'utilisation des ressources au sein de l'organisation ;
4. Les outils d'analyse et de modélisation de projets permettent de prendre des décisions réfléchies et adaptées, et de créer des rapports pertinents.

5.4. Synthèse sur la Gestion de Projet

Les procédés sont fortement utilisés dans le domaine de la gestion de projet, ils interviennent dans la partie planification et développement du projet. Nous remarquons que dans les procédés de gestion de projet le temps et les ressources (les personnes) sont des concepts centraux. En effet, le procédé est déplié par rapport au temps (pas de boucles dans le procédé), le procédé sert à gérer les ressources, et à coordonner le déroulement du projet dans le temps.

Les procédés, en général, peuvent donc être spécialisés suivant le concept central, sur lequel l'accent est mis. Nous avons alors :

- Les procédés centrés temps et ressources. Nous trouvons dans cette catégorie les procédés de gestion de projet ; et
- Les procédés centrés Activités, centrés Produits, et centrés Rôles. Nous trouvons dans ces catégories les procédés logiciels exécutables.

Nous trouvons que les procédés de gestion de projet et les procédés logiciels sont plus complémentaires que concurrents. C'est d'ailleurs pour cela que nous trouvons souvent plusieurs logiciels de gestion sur la même machine en même temps : un gestionnaire de projet, un gestionnaire de procédés, un gestionnaire d'espace de travail…

Nous considérons qu'il est important que ces divers environnements et outils puissent collaborer. Nous cherchons dans cette thèse à donner des solutions à ce problème.

[33] Extrait de [MS Project]

6. Orchestration et Chorégraphie de services web

Récemment, une nouvelle technologie nommée *services web* est apparue offrant un nouveau moyen pour échanger des données entre les applications. Tout logiciel peut dorénavant offrir ses fonctionnalités à travers ce nouveau moyen de communication qui utilise généralement le protocole HTTP (Hypertext Transfer Protocol [HTTP]).

Le développement d'applications à base de services web s'appuie sur la composition de ces derniers. Cela a relancé les travaux de recherche sur les problèmes d'interopérabilité et de coordination de services web. La solution proposée par la plupart de ces travaux consiste à modéliser le logiciel à réaliser par un procédé gérant la coordination des tâches et des opérations fournies par ce logiciel. Cela a donné naissance aux termes : *Orchestration* et *Chorégraphie*, et à la définition de plusieurs langages de coordination de services web.

Ce thème de l'orchestration et de la chorégraphie de services web étant un sujet central de cette thèse, nous le présentons de manière plus détaillée dans un chapitre à part (le chapitre III).

7. Synthèse générale

Nous avons présenté dans ce chapitre plusieurs domaines ayant utilisé les procédés de manière différente. Le tableau suivant (Table 4) synthétise les points importants que nous avons développé dans ce chapitre.

Domaine	Objectifs / Caractéristiques	Besoins non satisfaits
Procédés logiciels Contiennent les procédés : ▪ exécutables ; ▪ semi-formels ; ▪ organisationnels ; ▪ de gestion de configuration.	Gérer et assister le développement et l'évolution du logiciel. Le produit résultant de l'utilisation du procédé est le système logiciel. Trois catégories de PMLs de procédés logiciels exécutables (centrés activités, produits et rôles).	▪ simplifier les concepts pour modéliser les procédés ; ▪ permettre aux utilisateurs d'introduire leurs propres concepts, de manière simple et compréhensible.
CSCW	Coordonner les activités collaboratives. Le procédé fait partie du système logiciel, et participe à son exécution.	▪ permettre l'évolution du procédé ; ▪ permettre l'introduction de nouveaux concepts ; ▪ permettre l'évolution de concepts ; ▪ être simple et compréhensible, pour manipuler les concepts sans difficultés.
Workflow	Automatiser l'enchaînement des tâches dans les entreprises. ▪ Centré activité ; ▪ Simple ; ▪ Assez utilisé.	▪ l'interopérabilité entre WFMSs ; ▪ la reprise sur pannes ; ▪ des outils d'analyse, de test et de débogage ; ▪ besoin d'introduire de nouveaux concepts car les workflows sont trop simples (peu de concepts).
Gestion de projet	Planification et suivi du projet. Centrés ressources et temps.	La cohabitation des procédés de gestion de projet et des procédés logiciels dans un même système, car ils sont complémentaires.

Table 4 : synthèse générale

Chapitre III.
Etat de l'art sur l'orchestration et la chorégraphie de services web

1. Introduction

Nous avons introduit l'orchestration et la chorégraphie de services web dans la section 6 du Chapitre II. qui présente l'état de l'art des procédés en général.

Dans ce chapitre, nous allons présenter l'orchestration et la chorégraphie de manière plus détaillée. Nous allons commencer par la présentation d'une définition de l'orchestration et de la chorégraphie, en expliquant la différence entre ces deux termes. Nous allons ensuite détailler six langages/standards d'orchestration et de chorégraphie de services web, proposés depuis 2001. Ces six langages sont les plus connus dans le monde de la collaboration entre services web. Nous allons présenter ces langages développés par différentes entreprises en introduisant les concepts et une évaluation de chacun de ces langages.

Avant de présenter ces langages, il est important de connaître les bases de la technologie des services web. Pour cela, nous présentons brièvement dans cette introduction, le langage de description de services web : WSDL (Web Services Description Language), et le registre/annuaire global de services web : UDDI (Universal Description, Discovery and Integration).

1.1. WSDL - Web Services Description Language

WSDL est un langage de description de services web, utilisant le format XML[34]. Il décrit les services web comme un ensemble d'opérations et de messages abstraits.

Les éléments d'un service web [WSDL 01]

Le fichier WSDL décrit l'interface d'un service web, il contient les éléments suivants :

- **Types :** Contient les définitions de types décrits par un schéma XML [XSD] ;
- **Message :** Décrit les noms et types d'un ensemble de champs à transmettre (Paramètres d'une invocation, valeur du retour…) ;

[34] Extensible Markup Language [XML]

- **Opération :** Correspond à une abstraction décrivant une action implémentée par un service web ;
- **PortType :** Décrit un ensemble d'opérations. Chaque opération a zéro ou plusieurs message en entrée, zéro ou un message en sortie ou une faute (exception) ;
- **Liaison (binding) :** Spécifie une liaison d'un <PortType> à un protocole concret (SOAP 1.1[35], HTTP 1.1[36], MIME[37]…). Un PortType peut avoir plusieurs liaisons ;
- **Port :** Définit un point d'entrée (endpoint) comme la combinaison d'une <Liaison> et d'une adresse réseau ;
- **Service Web :** Est une collection de points d'entrée.

1.2. UDDI - Universal Description, Discovery and Integration

UDDI [UDDI] est un annuaire d'entreprises accessible par le web, il a pour but de permettre d'automatiser les communications entre prestataires et clients.

Les entreprises publient les descriptions de leurs services web dans l'annuaire UDDI sous forme de fichiers WSDL. Les clients peuvent ainsi rechercher plus facilement les services web dont ils ont besoin en interrogant le registre UDDI.

Lorsqu'un client trouve dans l'annuaire UDDI une description de service web qui lui convient, il télécharge son fichier WSDL depuis le registre UDDI. Ensuite, à partir des informations inscrites dans le fichier WSDL, notamment la référence vers le service web, le client peut invoquer les fonctionnalités du service web (Figure 19).

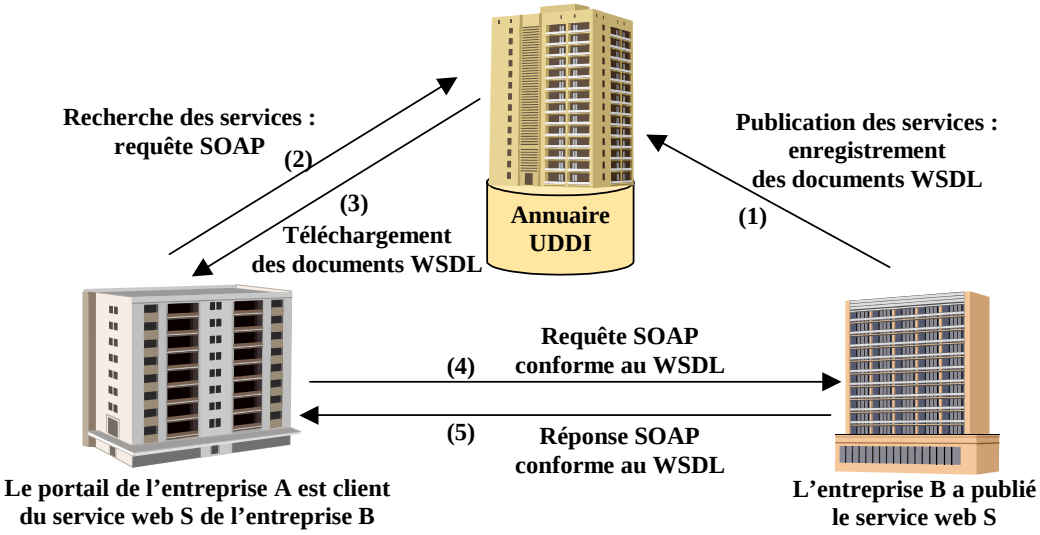


Figure 19 : Le protocole de découverte des services web via les annuaires UDDI [Ch 02]

[35] Simple Object Access Protocol [SOAP]

[36] Hypertext Transfer Protocol [HTTP]

[37] Multipurpose Internet Mail Extensions [MIME]

2. La définition de l'Orchestration et de la Chorégraphie

L'*orchestration* de services permet de définir l'enchaînement des services selon un canevas prédéfini, et de les exécuter à travers des "scripts d'orchestration". Ces scripts sont souvent représentés par des procédés métier ou des workflows inter/intra-entreprise. Ils décrivent les interactions entre applications en identifiant les messages, et en branchant la logique et les séquences d'invocation [SOE].
L'orchestration décrit la manière par laquelle les services web peuvent interagir ensemble au niveau des messages, incluant la logique métier et l'ordre d'exécution des interactions. Ces interactions peuvent couvrir des applications et/ou des organisations, et le résultat peut être un modèle de procédé de longue durée, transactionnel, et multi-étapes [Pel 03].

La *chorégraphie* trace la séquence de messages pouvant impliquer plusieurs parties et plusieurs sources, incluant les clients, les fournisseurs, et les partenaires. La chorégraphie est typiquement associée à l'échange de messages publics entre les services web, plutôt qu'a un procédé métier spécifique exécuté par un seul partenaire [Pel 03].

Il y a une différence importante entre l'orchestration et la chorégraphie de services web.

L'*orchestration* se base sur un procédé métier exécutable pouvant interagir avec les services web internes ou externes. L'orchestration offre une vision centralisée, le procédé est toujours contrôlé du point de vue d'un des partenaires métier.

La *chorégraphie* est de nature plus collaborative, chaque participant impliqué dans le procédé décrit le rôle qu'il joue dans l'interaction. Beaucoup de standards se sont intéressés initialement soit à l'orchestration soit à la chorégraphie. Mais les standards récents ont fusionné ces deux termes en un seul.

3. Langages d'orchestration et de chorégraphie de services web

Depuis la naissance des termes *Orchestration* et *Chorégraphie*, de nombreux langages de coordination de services web sont apparus. Certains de ces langages étaient plutôt centrés sur l'orchestration, d'autres plutôt sur la chorégraphie.

Dans cette section, nous allons présenter six langages d'orchestration et de chorégraphie de services web, que nous avons introduit dans le chapitre précédent. Ces six langages sont :

- XLANG - XML business process language
- BPML - Business Process Modeling Language
- WSFL - Web Services Flow Language
- WSCL - Web Services Conversation Language
- WSCI - Web Services Choreography Interface
- BPEL4WS - Business Process Execution Language for Web services

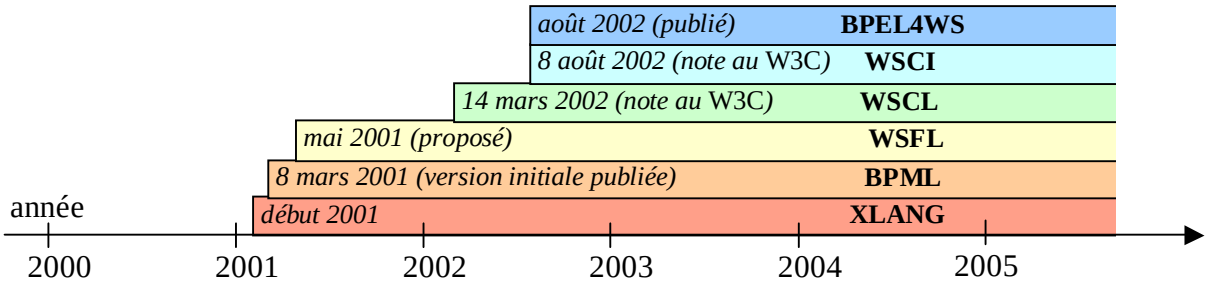


Figure 20 : Six langages d'orchestration et de chorégraphie de services web dans l'ordre chronologique de leur apparition

Dans ces six langages nous dégageons trois grands groupes :

Le premier groupe concerne les langages XLANG, WSFL, et BPEL4WS

Au début de l'année 2001, Microsoft a créé le langage XLANG, sous la forme d'extension de WSDL. En mai 2001, IBM a proposé le langage WSFL. Ces deux langages ont été ensuite remplacés par la spécification BPEL4WS, réalisée par IBM, Microsoft et BEA. Cette spécification a eu beaucoup de succès et est toujours utilisée.
Ces trois langages XLANG, WSFL, et BPEL4WS décrivent la coordination de services web.

Le deuxième groupe concerne le langage BPML

L'organisation BPMI.org a développé le langage BPML publié en mars 2001. Ce langage s'intéresse plutôt aux procédés métier (Business Process). Il gère la coordination de toute sorte de participants, et non pas de services web uniquement. Ce langage a également eu beaucoup de succès et est toujours utilisé.

Le troisième groupe concerne le langage WSCI

De leur coté, Sun, SAP, BEA, et Intalio ont proposé le langage WSCI, décrivant les interfaces de services web, pour réaliser une chorégraphie. Ce langage offre une vision différente de la collaboration, qui prend une forme de conversation entre plusieurs services web. La collaboration se passe d'une manière décentralisée, sans utiliser un procédé principal, mais plutôt plusieurs procédés distribués.

Un quatrième langage WSCL

Hewlett-Packard a soumis au World Wide Web Consortium (W3C), le 14 mars 2002, le langage de conversation WSCL. Ce langage ne s'occupe que de la description de conversations entre paires de services web. Il est donc plus léger que WSCI qui décrit des conversations entre plusieurs services web, et non pas seulement entre deux services web. Mais contrairement à WSCI, WSCL utilise une forme de procédé pour décrire la conversation, et l'exécution de la conversation n'est pas centralisée.

3.1. Le premier groupe : XLANG, WSFL, et BPEL4WS

Nous présentons dans cette section les trois langages XLANG, WSFL, et BPEL4WS.

3.1.1. XLANG - XML business process language [38]

Le langage XLANG est une extension de la spécification WSDL, créée par Microsoft et implémentée dans le produit BizTalk Server 2002.

Le processus complet ne s'exécute pas sur un moteur unique, il est constitué de la collaboration entre plusieurs sous processus éventuellement répartis sur des environnements d'exécutions hétérogènes.

1) Concepts du langage

Le fichier de description de service XLANG contient donc la description WSDL, et y ajoute les éléments suivants :

Le comportement (*behavior*)

Le comportement spécifie l'enchaînement des activités par des balises propres à XLANG. La partie comportement contient :

- Les actions XLANG : operation, delayFor, delayUntil, et raise ;+
- Le contrôle XLANG : qui définit l'enchaînement de l'exécution des actions, et qui peut être l'une des commandes : empty, sequence, switch, while, all, et pick.

Le contexte (*context*)

Le contexte spécifie les transactions éventuelles du processus et les références utilisées dans le document. Il permet de définir :

- des *variables locales*, constituées de *corrélations* et de *références* aux ports.
  Les corrélations sont des propriétés simples ou complexes, permettant de corréler les messages entre eux. L'ensemble des corrélations constitue une sorte d'index de messages ;

- des *transactions*.
  Les transactions étant longues, des compensations sont associées et exécutées lorsque la transaction est annulée. XLANG supporte à la fois les transactions ACID[39] et ACD[40], qu'elles soient réparties ou locales ;

- des *signaux* envoyés en cas de défaillance.

Le contrat (*contract*)

[38] Extrait de [Sat 01], de [KM 03] et de [Ch 02]

[39] ACID: Atomic, Consistent, Isolated and Durable Transactions [ACID]

[40] ACD : transactions non isolées, rend visible pour l'extérieur des modifications non encore validées [KM 03]

Le contrat spécifie les relations entre services XLANG.
Le contrat est spécifié dans un fichier à part. Ce fichier importe les fichiers XLANG des services participant à ce contrat, et il précise la manière dont les ports de ces services sont liés.

2) Systèmes exécutables basés sur ce langage

Microsoft a exploité le langage XLANG dans son environnement BizTalk [BizTalk]. BizTalk contient un environnement de développement : "BizTalk Orchestration Designer", permettant de générer les "schedules" XLANG (les fichiers XLANG sont appelés schedules). Le serveur BizTalk utilise ensuite ces "schedules" XLANG pour instancier les processus à exécuter.

3.1.2. WSFL - Web Services Flow Language [41]

Le langage WSFL est une proposition effectuée par IBM, concernant la composition de services web.

1) Concepts du langage

WSFL est un langage XML, pour décrire la composition de services web. Il permet de décrire :

- l'enchaînement des appels d'opérations de services web ;
- les interactions entre deux services web, en explicitant les relations "producteur/consommateur de messages" entre leurs descriptions WSDL. Cette description est abstraite, basée sur les deux fichiers WSDL, en laissant les services web "en blanc" afin de les implémenter plus tard.

La syntaxe WSFL

Dans le langage WSFL, six éléments spécifient une composition de services web :

- `"flowSource"` et `"flowSink"` indiquent les données en entrée et en sortie de la composition ;
- `"serviceProvider"` indique les services web participant à la composition ;
- `"activity"` décrit une opération ou une étape du dialogue ;
- `"controlLink"` et `"dataLink"` spécifient respectivement comment se succèdent les étapes et les données qui sont passées d'une étape à la suivante.

Le processus WSFL est donc une succession d'opérations dont le flot de contrôle et le flot de données sont explicités séparément.

L'implémentation de l'activité est soit déléguée à un service externe (par la balise `"Export"`), soit exécutée à partir d'autres activités du processus WSFL (indiqué par la balise `"Internal"`). Dans le cas de la délégation, le service externe est à son tour soit explicitement nommé, soit implicitement indiqué par un élément `"Locator"` identifiant le fournisseur de service.

[41] Extrait de [Ley 01] et de [Ch 02]

L'activité comporte des attributs facultatifs :

- L'attribut `"exitCondition" :` condition de sortie.
- L'attribut `"join" :` condition jointe.
- Ainsi qu'une condition de transition facultative, qui peut être ajoutée dans les liens de contrôle `"controlLink"`.

WSFL définit également des opérations de "contrôle du cycle de vie". Ces opérations lancent et contrôlent les activités :

- `"Spawn"` crée une instance de l'activité ou de la composition, et la démarre.
- `"Call"` exécute une instance d'activité ou de composition.
- `"Enquire"` renvoie l'état d'une activité ou d'une composition.
- `"Terminate"` arrête une instance d'activité ou de composition.
- `"Suspend"` suspend l'exécution en cours de l'instance.
- `"Resume"` reprend l'exécution d'une instance suspendue.

2) Systèmes exécutables basés sur ce langage

Le logiciel "MQ Series Workflow" de IBM, connu aujourd'hui sous le nom de "WebSphere Process Manager" [WS MQ], a pris en charge la spécification WSFL, afin d'automatiser les flots de procédés métier [Myerson].

IBM, Microsoft et BEA ont ensuite réalisé ensemble la spécification BPEL4WS, qui étend et remplace les précédentes spécifications XLANG de Microsoft, et WSFL d'IBM. Ce qui explique le peu d'implémentation de la spécification WSFL. BPEL4WS, en revanche, a eu beaucoup de succès, et beaucoup d'implémentations industrielles ont été créées pour prendre en charge cette spécification. IBM a même remplacé, dans le logiciel WebSphere, la spécification WSFL qu'il a pris en charge au début, par la spécification BPEL4WS, pour gérer les procédés métier (voir 3.1.3 section 2).

3.1.3. BPEL4WS - Business Process Execution Language for Web Services[42]

Le langage BPEL4WS est une spécification de IBM, Microsoft, et BEA. Elle remplace les précédentes spécifications XLANG de Microsoft, et WSFL d'IBM.

1) Concepts du langage

Le modèle de procédé BPEL4WS forme une couche au-dessus du WSDL. Il définit la coordination des interactions entre l'instance de procédé et ses partenaires.
BPEL4WS contient les caractéristiques d'un langage structuré en blocs (block structured language) du XLANG, ainsi que les caractéristiques d'un graphe direct de WSFL.

BPEL4WS permet de modéliser deux types de procédé :

[42] Extrait de [ACD 03]

- Le *procédé abstrait* : spécifie les échanges de messages entre les différentes parties, sans spécifier le comportement interne de chacun d'eux ;
- Le *procédé exécutable* : spécifie l'ordre d'exécution des activités constituant le procédé, des partenaires impliqués dans le procédé, des messages échangés entre ces partenaires, et le traitement de fautes et d'exceptions spécifiant le comportement dans les cas d'erreurs ou d'exceptions.

Les partenaires

La relation entre le procédé et un partenaire est une relation "poste à poste" (peer-to-peer). Le partenaire est en même temps le consommateur d'un service que le procédé produit, et le producteur d'un service que le procédé consomme.
Le lien de partenariat (partner link) définit le rôle que joue chacun des deux partenaires dans une conversation. Chaque lien de partenariat a un type (partnerLinkType).

Les propriétés de message

BPEL4WS ajoute des propriétés aux messages échangés entre partenaires. Un exemple de ces propriétés est la "corrélation", qui permet de faire le lien entre les messages appartenant à la même conversation.

Les activités

Le procédé dans BPEL4WS est constitué d'activités, liés par un flot de contrôle. Ces activités peuvent être *basiques* ou *structurées*.

Les activités basiques sont : *"invoke"* pour invoquer une opération dans un service web ; *"receive"* pour attendre un message d'une source externe ; *"reply"* pour répondre à une source externe ; *"wait"* pour attendre un certain temps ; *"assign"* pour copier les données d'une place à l'autre ; *"throw"* pour lancer une erreur d'exécution ; *"terminate"* pour terminer l'instance du service web ; et *"empty"* qui ne fait rien (utile pour la synchronisation des activités concurrentes).

Les activités structurées sont composées d'autres activités basiques et structurées. Les types d'activités structurées sont : *"sequence"* pour définir un ordre d'exécution ; *"switch"* pour l'acheminement conditionnel ; *"while"* pour les boucles ; *"pick"* pour attendre l'arrivée d'un événement ; *"flow"* pour l'acheminement parallèle ; *"scope"* pour regrouper les activités afin qu'elles soient traitées par le même gestionnaire d'erreur et *"compensate"* pour invoquer les activités de compensation par le gestionnaire d'erreur, pour défaire l'exécution déjà complétée d'un regroupement d'activité.

Les données

Le procédé dans BPEL4WS a un état, cet état est maintenu par des *"variables"* contenant des données. Ces données sont combinées afin de contrôler le comportement du procédé, pour cela nous utilisons les *"expressions"*. Les expressions permettent d'ajouter des conditions de transition ou de jointure au flot de contrôle.

L'affectation (*assignment*) permet de mettre à jour l'état du procédé, en copiant les données d'une variable à une autre, ou en introduisant de nouvelles données en utilisant les expressions.
Dans BPEL4WS il n'y a pas de flot de données, BPEL4WS se sert des variables pour passer une donnée d'une activité à une autre, à l'aide de l'affectation.

2) Systèmes exécutables basés sur ce langage

"Collaxa Inc." est une société, fondée en décembre 2000, établie sur l'idée que "Toute application est une orchestration". La vision de cette société est de se baser sur les standards d'orchestration pour construire les applications. Cette société était la première à construire un serveur d'orchestration de services web basé sur BPEL4WS. Les partenaires de Collaxa sont BEA Oracle, et Sun Microsystems [JS 02, Pel 03].

En juin 2004, Oracle a racheté Collaxa, afin de l'intégrer dans son serveur d'applications "Oracle Application Server 10g" [OP 04], ce qui en a fait un produit maîtrisant l'orchestration de procédé conformément à BPEL4WS [Spi 05].

Le laboratoire alphaWorks de IBM a produit BPWS4J (Business Process Execution Language for Web Services Java Run Time), disponible sur leur site web alphaWorks. BPWS4J permet de créer et d'exécuter des procédés métiers conformément à la spécification BPEL4WS [IBM 02, Pel 03].

IBM a remplacé dans WebSphere, la spécification WSFL, par cette nouvelle spécification BPEL4WS [KCH 04]. Microsoft a également introduit cette nouvelle spécification dans son produit BizTalk Server 2004, afin qu'il puisse importer et exporter des procédés métier écrits en BPEL4WS [Biz 04]. Ceci a été fait dans le cadre du projet "Jupiter".

La société Momentum [Momentum, Mom Perf] a également choisi BPEL4WS parmi les standards d'orchestration de services web, pour son produit d'automatisation de procédé ChoreoServer for .NET, conçu spécifiquement pour les vendeurs de logiciels indépendants et la redistribution commerciale. ChoreoServer [Choreo] a ensuite été transféré à la société OpenStorm [OpenStorm], qui se charge aujourd'hui de sa distribution. ChoreoServer est aujourd'hui disponible pour les plate-formes .Net et Java/J2EE .

3.1.4. Evaluation

Dans ce groupe, les langages de coordination utilisent un procédé centralisé pour décrire la coordination de services web.

Nous notons surtout le succès de BPEL4WS, à travers les nombreuses implémentations industrielles créées pour prendre en charge cette spécification. Ceci montre l'intérêt de cette approche basée sur un procédé de coordination centralisé.

3.2. Le deuxième groupe : BPML

Nous présentons dans cette section le langage BPML. Contrairement au premier groupe que nous avons présenté, ce langage s'intéresse plutôt aux procédés métier (Business Process). Il gère la coordination de toute sorte de participants, et non pas seulement de services web.

3.2.1. BPML - Business Process Modeling Language [43]

Le langage BPML a été développé par l'organisation BPMI.org *(Business Process Management Initiative)*, une organisation indépendante comprenant Intalio, Sterling Commerce, Sun, CSC, et d'autres.

1) Concepts du langage

Un processus métier BPML est un enchaînement d'activités simples, complexes, et de processus incluant une interaction entre participants dans le but de réaliser un objectif métier. Les éléments de définition d'un processus sont :

Les messages

Le message est l'unité d'interaction du processus. Les interactions de messages modélisent le stockage et la recherche de données, l'invocation de méthodes, et la gestion d'éléments de travail.

Les participants

Les participants peuvent être des systèmes, des applications, des services web, des utilisateurs humains, des partenaires commerciaux, et d'autres processus.
Il existe deux types de participants dans BPML : les participants statiques (où l'identité et le comportement sont connus à l'avance), et les participants dynamiques (ajoutés au moment de l'exécution).

Les activités

L'activité est l'unité d'exécution du processus. Les participants et les processus sont représentés en BPML comme des activités productrices et consommatrices de messages. Il existe deux classes d'activités dans BPML : les tâche simples, et les tâches complexes composées à partir de tâches simples.

Les transactions

L'activité peut posséder un attribut supplémentaire afin de spécifier si elle est exécutée dans un contexte transactionnel. Il est également possible de définir des compensations pour les transactions longues partielles. Les transactions dans BPML peuvent être emboîtées.
BPML définit deux modèles de transactions : coordonnées (pour les délais courts), et étendues (pour les délais longs).

Les exceptions

[43] Extrait de [Ark 02], de [Ch 02] et de [KM 03]

BPML définit un système de gestion d'exceptions, en ajoutant un dispositif de récupération d'erreur au moteur de procédé.

Les règles métier

Les règles du processus gouvernent le choix des tâches, la gestion de la consommation ou de la production de messages, et la réaction aux erreurs. Elles sont du style :

```
Si {conditions} alors {actions}
```

Les conditions portent sur les variables globales du processus ou sur les données échangées entre participants. Les actions déclenchent d'autres tâches BPML.

2) Systèmes exécutables basés sur ce langage

La société Intalio a construit, en mars 2001, le produit "Intalio|n³" qui est un système de gestion de procédés métier, basé sur le standard BPML, et non dépendant de la plate-forme d'exécution [Int n3 prod].

Intalio|n³ contient cinq composant permettant de concevoir, déployer, exécuter, maintenir, et optimiser le procédé métier. Le cœur de ce système, Intalio n³ Server, est une machine virtuelle de procédés, permettant d'exécuter les procédés métier conformément à la spécification BPML [Int n3 01].

Dans Intalio|n³, l'outil de conception permet de générer plus de 80% du code requit pour construire un procédé métier exécutable, ce qui réduit le temps de développement. L'environnement de gestion permet de gérer, configurer, et superviser les procédés déployés. Les utilisateurs peuvent interagir avec le procédé déployé à travers une interface de portail web [Pel 03].

3.2.2. Evaluation

Dans ce groupe, le langage BPML utilise également un procédé centralisé pour coordonner les participants au système. Ces participants ne sont pas que des services web, ils peuvent aussi être des applications, des utilisateurs humains, des partenaires commerciaux, et d'autres processus.

Ce langage a aussi eu beaucoup de succès, ce qui nous indique l'importance de prendre en compte la coordination de participants de différents types.

3.3. Le troisième groupe : WSCI

Dans ce groupe, nous décrivons le langage WSCI. Ce langage permet de spécifier les interfaces de services web, pour réaliser une chorégraphie entre eux. La collaboration se passe alors d'une manière décentralisée, sans utiliser un procédé principal, mais plutôt plusieurs procédés distribués.

3.3.1. WSCI - Web Services Choreography Interface [44]

Le langage WSCI est une Initiative proposée par Sun, SAP, BEA, et Intalio. L'objectif consiste à prendre en compte la collaboration d'application à application. Cette initiative s'est concrétisée par une note au World Wide Web Consortium (W3C), le 8 août 2002.

1) Concepts du langage

WSCI est un langage XML, permettant de décrire la chorégraphie entre les services web. Ainsi, les interfaces statiques des services web sont décrites en WSDL, et la chorégraphie entre eux est décrite en WSCI. Le langage WSCI contient les concepts suivant :

L'interface

Le but de WSCI est de décrire les détails du comportement d'un service web, en terme de dépendances temporelles et logiques entre les messages que ce service web échange avec d'autres services web, dans le contexte d'un scénario. Ce comportement est décrit dans un ou plusieurs processus contenus dans l'interface. Un service web peut avoir plusieurs interfaces lui permettant de jouer différents scénarios.

Les activités et leur chorégraphie

WSCI décrit le comportement d'un service web en terme d'activités chorégraphiés. La chorégraphie décrit les dépendances temporelles et logiques entre les activités.

Les activités peuvent être atomiques (activité d'envoie et/ou de réception d'un message, ou activité d'attente d'une durée définie) ou complexes (composées d'autres activités). L'activité complexe définit la chorégraphie des activités dont elle est composée. Plusieurs types de chorégraphie existent dans WSCI : l'exécution séquentielle, l'exécution parallèle, la boucle, et l'exécution conditionnelle.

Les processus

Le processus contient une partie du comportement du service web, il représente l'unité de réutilisation dans WSCI. Deux types de processus sont définis dans WSCI : les processus définis au niveau de l'interface, et les processus définis à l'intérieur des activités complexes.

Les propriétés

Les propriétés dans WSCI sont l'équivalent des variables dans les langages de programmation. Les propriétés sont utilisées pour éviter, dans le fichier WSCI, les références explicites vers les messages abstrait décrits en WSDL.

Le contexte

Le contexte décrit l'environnement dans lequel s'exécute un ensemble d'activités. Chaque activité est définie dans un et un seul contexte.

[44] Extrait de [AAF 02]

Le contexte décrit l'environnement d'exécution en terme : des déclarations (propriétés locales ou définitions locales de processus) disponibles pour les activités, des événements d'exception qui peuvent arriver, les propriétés transactionnelles associées à l'exécution des activités. Les contextes peuvent être emboîtés.

La corrélation de messages

Dans WSCI, une conversation représente un échange de messages entre deux ou plusieurs services web, participant à un scénario. Un service web peut être engagé dans plusieurs conversations en même temps, avec le même service ou avec des services différents.

La corrélation est le mécanisme par lequel un message, reçu par le service, est associé à une conversation particulière. Ainsi, les différentes conversations sont distinguées par les instances de corrélation (une instance de corrélation est un ensemble de valeurs de propriétés).

Les exceptions

WSCI permet de déclarer des exceptions dans la définition de contexte, et de définir un ensemble d'activités que le service web aura à exécuter lorsque cette exception se produit. Les exceptions déclarées peuvent être : la réception d'un message d'exception, la production d'une erreur ou le dépassement de la date limite de fin.

Ainsi, lorsqu'une exception se produit, cela n'arrête pas la chorégraphie en entier, mais seulement le contexte où l'exception s'est produite après avoir exécuté les activités spécifiques à cette exception. Si le contexte ne contient pas de déclaration d'exceptions, l'exception est remontée au contexte "parent".

Les transactions

WSCI permet d'associer une transaction dans le contexte. Ainsi, les activités contenues dans cette transaction seront exécutées de la manière "tout ou rien". La transaction peut contenir un ensemble d'activités de compensation[45], exécutées s'il y a besoin de défaire la transaction une fois qu'elle est terminée. Les transactions sont soit atomiques, soit emboîtées (composées d'autres transactions).

2) Systèmes exécutables basés sur ce langage

Sur Microsystems a réalisé un éditeur d'interfaces WSCI : le Sun ONE WSCI Editor. Une version alpha de cet éditeur a été publiée en accès gratuit. Cette version de l'éditeur est assez légère et ne fournit que les fonctionnalités basiques pour construire des fichiers d'interface WSCI, à partir de fichiers WSDL. Ces interfaces ne peuvent pas être testées, car les interfaces WSCI ne sont pas exécutables par définition [Pel 03].

[45] Dans le contexte des transactions, une compensation est une action qui s'exécute lorsque la transaction ne se termine pas correctement, et rend l'état du système incohérent. La compensation a alors pour but de réparer cette incohérence en annulant les modifications effectuées, et en remettant ainsi le système dans l'état cohérent précédent.

3.3.2. Evaluation

Dans ce groupe, le langage WSCI permet de décrire les interfaces de services web pour réaliser une chorégraphie entre eux. La collaboration se passe alors d'une manière décentralisée en utilisant plusieurs procédés distribués plutôt qu'un procédé principal.

Les interfaces WSCI sont non exécutables. Et il n'y a, à ce jour, aucun système permettant d'exécuter une chorégraphie basée sur les interfaces décrites en WSCI. Ceci est sûrement dû à la difficulté de réaliser une coopération de services web de manière décentralisée, et aux inconvénients que cela apporte, comme par exemple le manque de vision globale de la coopération.

3.4. Un quatrième langage WSCL

Le langage WSCL apporte une approche légèrement différente des précédentes, car il se concentre sur la description de conversations entre paires de services web. Il est donc plus léger que WSCI.
WSCL utilise un procédé pour décrire la conversation et son exécution est décentralisée.

3.4.1. WSCL - Web Services Conversation Language [46]

Le langage WSCL est une soumission de Hewlett-Packard au World Wide Web Consortium (W3C), sous forme d'une note, le 14 mars 2002. L'objectif consiste à décrire la séquence d'interactions possibles avec un service web particulier.

1) Concepts du langage

WSCL est un langage XML, il permet de représenter simplement les interactions entre deux services web.

La spécification WSCL est composée de quatre éléments principaux :

- Les *schémas* des documents XML échangés au cours d'une conversation. Ces schémas ne font pas partie du document de spécification WSCL, ce sont des documents séparés que l'on référence par un URL (Uniform Resource Locator) dans la spécification de la conversation ;
- Les *interactions* modélisant les actions de la conversation comme des échanges de documents entre deux participants. Il existe cinq types d'interactions dans WSCL :
  - "Send" : émission d'un message,
  - "Receive" : réception d'un message,
  - "SendReceive" : émission puis réception d'un message,
  - "ReceiveSend" : réception puis émission d'un message,
  - "Empty" : ne contient pas de message à échanger, il est utilisé pour modéliser le début et la fin d'une conversation,

[46] Extrait de [WSCL 02] et de [Ch 02]

- Les *transitions* spécifiant l'ordonnancement les relations entre les interactions. Chaque transition spécifie l'interaction source et l'interaction de destination, et éventuellement le type de message de l'interaction source ;
- La *conversation* donne la liste de toutes les interactions et les transitions composant la conversation, ainsi que quelques informations additionnelles, comme le nom de la conversation, et l'interaction qui démarre la conversation, et celle qui la termine.

WSDL se charge de décrire les services web, et WSCL se charge de décrire les conversations entre deux services web. Ainsi, ces deux descriptions sont séparées, afin de permettre la réutilisation (une même conversation WSCL peut avoir lieu entre différentes paires de services web décrites en WSDL, et un service web décrit en WSDL peut participer à plusieurs conversations WSCL). Cette approche différencie WSCL des autres approches, tels que XLANG et WSFL.

Un fournisseur de service peut soit fournir la définition de conversation WSCL directement à l'utilisateur du service, soit l'enregistrer en tant que des "tModel" dans un annuaire UDDI [Ari 00, KLB 01], pour qu'elle soit à disposition de tous ceux qui ont accès à cet annuaire.

2) Systèmes exécutables basés sur ce langage

Il n'y a pour l'instant, à ma connaissance, aucun système exécutable basé sur le langage WSCL.

3.4.2. Evaluation

Le langage de conversation WSCL ne s'occupe que de la description de conversations entre paires de services web. Il est donc plus léger que WSCI qui décrit des conversations entre plusieurs services web, et non pas seulement entre deux services web.

WSCL a utilisé les procédés dans la description de conversation. Mais l'exécution de la conversation, suivant le langage WSCL, est décentralisée.

Comme dans le cas de WSCI, aucun système permettant d'exécuter une conversation basée sur WSCL n'existe. Ce qui confirme la difficulté de réaliser une coopération décentralisée de services web.

Toutefois, WSCL offre une vision globale de la conversation en la décrivant par un procédé. Ce qui est un avantage par rapport à WSCI.

4. Synthèse sur l'orchestration et la chorégraphie de services web

Les langages que nous avons présentés dans ce chapitre sont les plus connus dans le monde de la collaboration entre services web. Ces langages ont des similitudes ainsi que des différences. Nous présentons dans cette section les points importants concernant ces langages.

4.1. Orchestration ou Chorégraphie

Le principe de certains de ces langages, comme BPML, est de construire un procédé centralisé, se chargeant de coordonner les services web. Ces langages-là font donc de l'orchestration.

D'autres langages, comme WSCL, ne s'occupent que de la description de "conversations" entre les paires de services web. Autrement dit, il ne se charge que de décrire les interactions entre chaque couple de services web, sans spécifier comment est créé le contenu des messages échangés. Cette description de conversation prend une forme de procédé, mais son exécution n'est pas centralisée. La collaboration des deux services web entre donc, dans ce cas-là, dans le domaine de la chorégraphie.

Dans des langages comme XLANG et WSCI, chaque service web connaît son comportement, et la réaction qu'il aura suite à chaque opération et à chaque message arrivant. Le langage offre en effet une description du comportement du service web, formant une couche au dessus du fichier WSDL, décrivant les opérations et les messages de ce service web. Dans WSCI, ceci est fait dans la description des interfaces de services web uniquement. XLANG, par contre, ajoute à cette description d'interfaces un contrat spécifié dans un fichier à part, précisant les liaisons entre les ports de ces services.
Dans les deux cas, la collaboration entre services web se passe d'une manière décentralisée. Ces langages font donc de la chorégraphie.

Il y a aussi des langages, comme BPEL4WS, qui permettent de décrire deux types de collaboration :

- La coordination centralisée, lorsqu'il s'agit de décrire l'enchaînement des appels d'opérations de services web (le "procédé exécutable" dans le langage BPEL4WS). Ce qui correspond à faire de l'orchestration ;
- Et la collaboration décentralisée, lorsqu'il s'agit de décrire le contrat d'interactions entre deux services web, ou ce qu'on a appelé "la conversation" (le "procédé abstrait" dans BPEL4WS). Ce qui correspond à faire de la chorégraphie.

Ces langages font donc de l'orchestration et de la chorégraphie.

4.2. Représentation et édition

Tous ces langages décrivent la collaboration entre services web sous forme de fichiers XML. Ces fichiers XML sont souvent très difficiles à lire et à manipuler. Il est également difficile de garantir la cohérence et la correspondance entre les différentes parties éparpillées sur les nombreux fichiers XML.

Pour cela, les entreprises utilisant ces langages ont créé des éditeurs graphiques, afin de faciliter la tâche d'édition de fichiers de description de la collaboration.

4.3. Exécution

Les langages que nous avons présentés précisent les concepts de la collaboration entre services web, mais ils ne précisent pas la manière dont ces concepts doivent être implémentés. Des systèmes exécutables, basés sur ce langage, ont été construit par des entreprises, afin de permettre l'exécution de la collaboration construite par ces langages. Nous avons présenté quelques-uns de ces systèmes.

Nous remarquons qu'il y a eu beaucoup d'implémentations industrielles basées sur le langage BPEL4WS, qui est une évolution de XLANG et WSFL. Ceci indique que les concepts de ce langage correspondaient au besoin de beaucoup d'entreprises.

Nous remarquons également qu'il n'y a pas eu d'implémentation permettant d'exécuter une collaboration décrite dans le langage WSCI. Il n'y a eu qu'un éditeur permettant de faciliter la rédaction des interfaces non-exécutables décrite en WSCI. Il n'y a pas eu non plus d'implémentation du langage WSCL. Ce qui indique soit la difficulté d'implémentation, soit le peu d'intérêt à décrire les interfaces de chorégraphie par rapport à l'orchestration.

Certaines entreprises ont également construit des logiciels de supervision, afin de suivre l'exécution de la collaboration. Notamment plusieurs implémentations du langage BPEL4WS ont ajouté la supervision de l'exécution à leurs produits. La supervision de l'exécution est, de notre point de vue, un point essentiel pour la réussite du système d'exécution de la collaboration.

4.4. Vision globale de la collaboration

Dans l'orchestration, un procédé centralisé est défini afin de coordonner les services web. Ceci permet de donner une vision globale de la coordination.

Dans la chorégraphie, la collaboration entre services web est décentralisée. Les langages comme XLANG ou WSCI expriment les interactions que peut avoir chacun des services web avec les autres services. Par contre, ils ne fournissent pas l'ensemble des interactions entre tous les services web. Nous ne pouvons donc pas, avec ces langages-là, obtenir une vision globale de la collaboration.

WSCL, WSFL et BPEL4WS, faisant aussi de la chorégraphie, un contrat d'interactions entre les deux services web est décrit. Ce contrat, décrivant la conversation, forme un procédé abstrait, qui donne une vision globale de la collaboration entre les services web.

Dans les deux styles de collaboration (l'orchestration et la chorégraphie), le procédé a une place importante car il fournit une vision globale de la collaboration. La présence du procédé dans la plupart des langages de collaboration témoigne de cette importance. Comme nous l'avons vu, le procédé est absent uniquement chez WSCI, qui n'offre que des interfaces non-

exécutables, et chez XLANG, qui à été remplacé par BPEL4WS, contenant deux sortes de procédés : abstrait et exécutable.

4.5. Participant à la collaboration

La plupart de ces langages ne gèrent que la collaboration de services web, comme XLANG, WSFL, WSCL, WSCI, BPEL4WS. Mais certains langages, comme BPML, gèrent la coordination de toute sorte de participants : des systèmes, des applications, des services web, des utilisateurs humains, des partenaires commerciaux, et d'autres processus.

Nous avons vu dans la section 3.1.3 que Collaxa a été intégré dans le serveur d'applications d'Oracle, pour faire de l'orchestration de procédé, conformément à BPEL4WS. Dans ce produit, le procédé BPEL4WS peut interagir avec les services web, mais aussi avec les JCA[47], JMS[48], workflows, et les e-mails [Spi 05].

De notre point de vue, le fait de restreindre la collaboration aux services web est une limitation importante. Nous prônons un système de collaboration général, permettant de faire collaborer différents participants (des services web, comme des logiciels patrimoniaux /legacy systems/, des applications locales ou distantes, des humains, des processus…).

4.6. Description de la collaboration et fichiers WSDL

Les langages ne gérant que la collaboration de services web, comme XLANG, WSFL, WSCL, WSCI et BPEL4WS forment souvent une sur-couche aux fichiers WSDL. Il existe deux façons de réaliser cette sur-couche :

- En étendant le fichier WSDL en lui ajoutant les informations de coordination, comme dans le cas du langage XLANG ;
- En séparant les informations additionnelles dans un fichier à part, collaborant avec le fichier WSDL, c'est le cas par exemple du langage BPEL4WS.

Certains de ces langages (comme WSFL et BPEL4WS) proposent de créer des modèles de composition récursifs. Une composition de services web écrite en WSFL ou en BPEL4WS peut être considérée comme un nouveau service web que l'on peut utiliser dans une nouvelle composition.
Cette composition récursive sert également à faire, de manière indirecte, de la composition hiérarchique.

Cette forme de collaboration, limitée aux services web est, comme nous l'avons souligné dans le paragraphe précédent, très restrictive.

[47] JCA : J2C Connector Architecture

[48] JMS : Java Message Service

De plus, l'expression de la collaboration fortement liée aux fichiers WSDL, constitue une forme d'expression de bas niveau, se rapprochant beaucoup d'un langage de programmation. Nous allons détailler cette vision dans le paragraphe suivant.

4.7. Langages de coordination vs. langage de programmation

Dans l'étude que nous avons faite sur les langages de coordination de services web, nous avons vu que ces langages sont très proches de la programmation. Ecrire une coordination avec un de ces langages est assez similaire au fait de la programmer.

Nous constatons cela surtout dans la correspondance "1 à 1" entre l'activité du procédé écrit dans un langage de coordination et l'opération du service web qui l'implémente. Nous le constatons également dans la rigidité du lien entre l'activité et son implémentation, qui, dans les langages de coordination, est exprimé "en dur" dans le fichier de description de la coordination. Ce qui est équivalent à un appel de méthode dans un langage de programmation. Le procédé de coordination n'est donc rien qu'une succession d'appels à des opérations de services web.

Cette rigidité du lien entre l'activité et son implémentation pose des problèmes lorsqu'il faut faire évoluer le modèle de collaboration (modifier le modèle de collaboration, y ajouter de nouvelles opérations, supprimer des opérations anciennes…). Cela pose également des problèmes lorsqu'il y a besoin de substituer une fonctionnalité d'un service web par une autre. La réutilisation du modèle de procédé est d'autant plus difficile.

4.8. Quelques travaux de recherche

Van der Aalst a définit 20 patterns pour évaluer les workflows [VTK 02, VdA patt], et a même proposé un nouveau langage de workflow, YAWL, basé sur ces patterns [VT 02, VT YAWL]. Il considère que ces 20 patterns identifient les fonctionnalités des workflows. Ces patterns forment donc, suivant sa vision, une mesure permettant d'évaluer les fonctionnalités des systèmes de gestion de workflows.

Van der Aalst a utilisé ses 20 patterns, pour évaluer les langages de collaboration de services web [WVD 02, VDT 02]. Sa vision des langages de collaboration rejoint ainsi la notre, en disant que ces langages sont similaires aux systèmes de gestion de workflows, car ils s'intéressent à construire des procédés exécutables.
Dans son évaluation des langages de collaboration de services web, Van der Aalst a utilisé également 6 patterns de communication présentés dans [RMB 01]. Il fournit des tableaux de comparaison entre ces langages de collaboration de services web, utilisant les patterns de workflow, et les patterns de communication.

Dans nos travaux autour des systèmes de gestion de workflows et des langages de collaboration de services web, nous avons une perception plus globale de ces systèmes. Nous

ne jugeons pas un système de gestion de procédé par rapport aux fonctionnalités diverses que peut offrir son flot de contrôle. Car nous estimons que ces fonctionnalités diverses, même si elles ne figurent pas explicitement parmi les fonctionnalités offertes par ce système, peuvent être effectuées en combinant les fonctionnalités basiques. En revanche, nous jugeons un système de gestion de procédé en fonction de sa capacité d'apporter de la facilité de construction, d'évolution, de supervision, et de compréhension, pour un utilisateur de niveau moyen. C'est en cela que nos travaux se diffèrent de ceux de Van der Aalst.

5. Conclusion

Nous avons présenté, dans ce chapitre, plusieurs langages d'orchestration et de chorégraphie de services web. Chacun de ces langages a apporté de nouveaux concepts, et a redéfini certains concepts déjà connus par les autres langages. Ceci a créé une multitude de concepts, se chevauchant parfois, ainsi qu'une multitude de façons de les manipuler.

Ces concepts, définis par les langages d'orchestration et de chorégraphie, sont intéressants, et apportent des compétences et des connaissances additionnelles dans le monde de la coordination de services web. Cependant, ces compétences sont parfois applicables dans un domaine particulier et pour des besoins particuliers qui ne sont pas toujours nécessaires dans les autres domaines.

Regrouper tous ces concepts dans un seul et unique langage nous paraît utopique, et ceci pour deux raisons. D'une part, car un méta-modèle d'orchestration unique et complet sera un méta-modèle extrêmement complexe. Et d'une autre part, car ce méta-modèle sera impossible à réaliser, car il y aura sûrement de nouveaux concepts à intégrer pour réaliser de nouveaux scénarios et pour répondre à de nouveaux besoins particuliers.

Nous avons également vu, dans la synthèse sur l'orchestration et la chorégraphie de services web que nous avons présentée (section 4), que la plupart de ces langages ne gèrent que la collaboration de services web (section 4.5), et que ces langages sont très proches de la programmation, fortement liés aux services web et difficilement modifiables (section 4.7).

Nous pouvons alors résumer les défauts des langages d'orchestration et de chorégraphie par les cinq points suivants :

- beaucoup de concepts spécialisés ;
- les concepts se chevauchent ;
- souvent réservé aux services web ;
- procédé et services web trop fortement couplés ;
- rigide (difficilement modifiable) ; et
- bas niveau (proche des langages de programmation).

Dans cette thèse nous cherchons à donner des solutions à ces défauts, ainsi qu'aux défauts des autres domaines utilisant les procédés, que nous avons présenté dans notre état de l'art, et que nous synthétisons dans la section suivante (section 6).

6. Synthèse générale de l'état de l'art

Tous les domaines présentés dans notre état de l'art (Chapitre II et Chapitre III) ont utilisé les procédés de manière différente. Nous avons remarqué dans ces domaines certains défauts, concernant les procédés de coordination, et d'autres concernant les systèmes de gestion de coordination existants. Le tableau suivant (Table 5) résume ces défauts, et présente les besoins qu'il faut prendre en compte dans les systèmes de gestion de coordination par procédés.

Défauts	Besoins
Procédés de Coordination	
Certains procédés ne sont pas exécutables	Procédés exécutables de coordination, centré activités
▪ Complexité des concepts ; ▪ Beaucoup de concepts spécialisés ; ▪ Les concepts se chevauchent.	▪ Simplifier les concepts ; ▪ Se baser sur un minimum de concepts ; ▪ Permettre l'évolution, en permettant l'introduction de nouveau concepts.
Fortement couplée (outils fortement liés au procédé)	Couplage flexible (facilité de substitution d'un service par un autre)
Rigide (difficilement modifiable)	Procédé flexible (facile à changer et à faire évoluer)
Bas niveau (proche des langages de programmation).	Formalisme de haut niveau (décrit la logique de l'application)
Coordination souvent réservée à des outils particuliers, ou à une technologie particulière, spécifique au domaine (exemple : dans le domaine de l'orchestration, souvent réservé aux services web)	Coordonner des outils divers (services web, programmes locaux ou distants, composants, COTS, etc.)

Défauts	Besoins
Systèmes de Gestion de Coordination	
Dans les systèmes existant : ▪ Performance insuffisante ; ▪ Ne permet pas le reprise sur pannes ; ▪ Manque d'outils d'analyse, de test, et de débogage ; ▪ Manque de prise en charge de	Environnement : ▪ Performant ; ▪ Gérant la reprise sur pannes ; ▪ Contenant des outils d'aide pour l'utilisateur (outils d'analyse, de test, et de débogage) ;

l'interopérabilité entre le système et les outils.	▪ Gérant l'interopérabilité entre le système et les outils.
Plusieurs logiciels de gestion sur la même machine ne communiquant pas.	Un logiciel spécialisé s'occupant de l'ensemble de nos besoins afin de réaliser une gestion cohérente et unifiée.

Table 5 : défauts et besoins des procédés de coordination, et des systèmes de gestion de coordination

Dans cette thèse, nous cherchons à répondre à ces besoins, afin de résoudre les défauts que nous avons identifiés. Pour cela nous proposons un environnement permettant de construire des logiciels ayant une architecture particulière. Cette architecture nous permet de résoudre beaucoup de problèmes que nous trouvons dans les logiciels de coordination à l'aide de procédé, comme les problèmes de substitution, et de facilité la modification du procédé, ainsi que l'utilisation d'un formalisme de haut niveau pour décrit la logique de l'application. Nous verrons, dans le Chapitre IV, les choix que nous avons pris pour définir cette architecture et pour réaliser notre environnement.

Nous allons voir également, comment nous avons réussi à répondre au besoin concernant la construction d'un logiciel s'occupant de l'ensemble de nos besoins, et couvrant plusieurs domaines de compétence tout en gardant un nombre minimal de concepts simples. Nous expliquons cela dans notre approche que nous présentons dans le Chapitre V.

Chapitre IV.

Notre plate-forme Mélusine

1. Introduction

Dans la section 2.1.3 du Chapitre II. nous avons présenté "APEL" (Abstract Process Engine Language) [EDA 98, ECB 98, ED 96, EAD 99], le PML de procédé logiciel exécutable développé dans notre équipe. APEL est l'évolution du système ADELE (1988) que nous avons également présenté dans le Chapitre II. (voir ADELE section 2.1.2).

L'objectif du travail de notre équipe autour d'APEL était de fournir un environnement de support complet pour la gestion des procédés logiciels. La première version, APEL V1 (1992), était basée sur des mécanismes de déclencheurs (trigger). Ces mécanismes-là étaient efficaces, mais difficiles à manipuler. Ainsi, APEL a été modifié en ajoutant un formalisme de haut niveau pour cacher les triggers. Ce qui a donné lieu à la deuxième version, APEL V2 (1993). Une troisième version APEL V3 a été réalisé en 1995 et une quatrième version APEL V4 en 1997. Cette quatrième version offrait un environnement de support des procédés logiciels très riche en concepts et monolithique, mais cette richesse de concepts l'a rendu très complexe, et difficile à manipuler, et son monolithisme a rendu son évolution difficile.

Au début des années 2000, un travail de simplification du meta-modèle d'APEL a été effectué. Ces travaux ont abouti à la cinquième version APEL V5 qui est un noyau minimal ne contenant que les concepts principaux.

Au même moment, la notion de Fédération a commencé à faire son apparition dans notre équipe. Une Fédération peut être définie comme étant : "une architecture logicielle qui matérialise la coordination, et qui permet de structurer les applications comme un ensemble de mondes qui coopèrent pour atteindre un but commun" [Vil 03].

Au cours de ces dernières années, un travail important a été mené autour des Fédérations par plusieurs personnes de notre équipe, dont je fais partie. Cela s'est concrétisé par plusieurs thèses réalisées dans notre équipe [Vil 03, Le 04, Les 03] ainsi que de plusieurs thèses en cours.

Ces travaux de recherche ont donné naissance a notre plate-forme Mélusine, qui est un environnement de conception, d'implémentation et d'exécution de logiciels basés sur cette architecture.

Nous avons intégré APEL[49] dans notre plate-forme Mélusine, afin de l'utiliser pour la construction d'applications à base de procédés, ayant une architecture logicielle de Fédération.

Dans le cadre de mon travail sur les Fédération, j'ai participé au projet industriel Centr'Actoll [Centr'Actoll]. Ma participation a consisté à utiliser Mélusine pour réaliser une Démo Métier dont l'objectif était d'enrichir le site web d'AREA (péage autoroutes Rhône-Alpes) par de nouvelles fonctionnalités. Ce projet nous a montré l'importance de faire évoluer certaines idées autour de l'extension d'applications. Nous avons continué, par la suite, à nous intéresser à l'extension d'applications. Nous avons élaboré ces idées grâce à nos travaux de recherche et aux applications que nous avons implémentées. Nous présentons nos travaux sur l'extension d'applications dans le chapitre suivant.

Dans mon travail de recherche, je me suis intéressée plus particulièrement aux procédés. Je considère qu'il est essentiel d'intégrer le procédé dans la construction d'applications car cela apporte de la clarté au niveau de la formalisation de l'enchaînement des tâches de l'application. Je me suis ainsi intéressée à la construction d'applications à l'aide de Mélusine, en utilisant APEL pour coordonner les tâches de l'application. J'ai alors implémenté une application d'orchestration de services web en utilisant la plate-forme Mélusine pour avoir une architecture logicielle de Fédération et en utilisant APEL pour coordonner les services web. J'ai utilisé ensuite nos travaux sur l'extension que je présente dans le chapitre suivant, pour étendre cette application.

Dans ce chapitre, nous présentons notre plate-forme Mélusine. Nous illustrons notre description de Mélusine à partir d'exemples utilisant APEL. Nous présentons ensuite la Démo Métier réalisée dans le cadre du projet industriel Centr'Actoll. Nous terminons par la présentation de l'application d'orchestration de services web, réalisée en utilisant la plate-forme Mélusine et APEL.

2. Notre plate-forme Mélusine

Comme nous l'avons mentionné dans l'introduction, la plate-forme Mélusine est un environnement de conception, d'implémentation et d'exécution de logiciels basés sur une architecture logicielle particulière que nous appelons une fédération.

La conception de logiciels à l'aide de la plate-forme Mélusine se fait en deux étapes. La première étape consiste à construire la Machine Virtuelle du Domaine d'Applications. Et la deuxième étape consiste à construire l'Application du Domaine, à l'aide de cette Machine Virtuelle.

[49] Dans la suite de ce document, nous utiliserons le terme APEL pour faire référence à la dernière version APEL V5.

2.1. La Machine Virtuelle du Domaine d'Applications

Un Domaine d'Applications peut être défini comme étant un ensemble de systèmes construits à partir de concepts communs [EIV 05].

Pour concevoir un logiciel à l'aide de la plate-forme Mélusine, il faut tout d'abord définir le méta-modèle du domaine d'applications. Ensuite, conformément à ce méta-modèle, il faut élaborer la Machine Virtuelle du Domaine d'Applications pour le logiciel que nous souhaitons concevoir.

Le méta-modèle du domaine d'applications permet de définir les concepts métier du domaine de notre application, et de préciser les relations entre ces concepts. Selon la philosophie de Mélusine, le méta-modèle doit être minimal et ne contenant que les concepts métier essentiels formant les éléments de base.

La Machine Virtuelle du Domaine d'Applications est un moteur permettant d'exécuter les modèles d'applications construits conformément au méta-modèle du domaine d'applications que nous avons défini.

Une fois que la Machine Virtuelle du Domaine d'Applications est réalisée, nous pouvons alors construire l'application du domaine, ayant une architecture logicielle de Fédération. Cette application est basée sur un modèle, que nous construisons conformément au méta-modèle du domaine d'applications, et qui sera exécuté à l'aide de notre Machine Virtuelle du Domaine d'Applications.

2.1.1. APEL : la Machine Virtuelle du Domaine de Procédés

Comme nous l'avons déjà mentionné, nous nous sommes intéressés dans ce travail de thèse aux procédés et plus particulièrement à la construction d'applications dirigées par les procédés. Une application dirigée par un procédé est une application basée sur un procédé de coordination qui formalise l'enchaînement des tâches de l'application.

Comme nous venons de le voir, pour réaliser de telles applications à l'aide de Mélusine, il nous faut disposer d'un méta-modèle de procédé et d'une Machine Virtuelle du Domaine de Procédés. Nous avons tout naturellement profité de l'expérience acquise dans notre équipe dans le domaine des procédés pour construire un méta-modèle de procédé conforme à nos exigences (le méta modèle d'APEL) ainsi qu'une Machine Virtuelle correspondante permettant d'exécuter les modèles de procédés conformes à ce méta-modèle (le moteur APEL).

Nous présentons dans cette section ce méta-modèle de procédé d'APEL, ainsi que l'éditeur graphique d'APEL, permettant de construire des modèles de procédés suivant ce méta-modèle.

1) Le méta-modèle de procédé

APEL offre un méta-modèle de procédé conforme à la philosophie de Mélusine. En effet, ce méta-modèle ne contient qu'un minimum de concepts formant les éléments de base. Ces concepts sont : les activités, les produits, leur types, les rôles, les ressources, les flots de données et les ports. La Figure 21 reprend le méta-modèle de procédé d'APEL que nous avons présenté dans la Figure 7 de la section 2.1.3, Chapitre II.

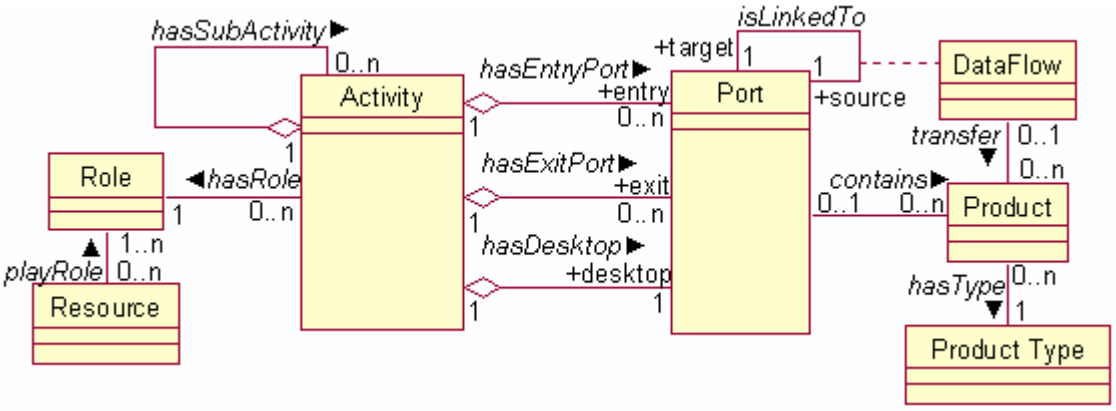


Figure 21 : le méta modèle d'APEL

- Une *activité* est une étape du procédé au cours de laquelle une action est exécutée ;
- Un *produit* est une donnée produite, transformée et consommée par les activités ;
- Les *ports* sont les interfaces de l'activité, ils définissent et contrôlent la communication entre l'activité et le monde extérieur ;
- Un *rôle* est une qualification associée à une ou plusieurs personnes. Une activité à un responsable (une *ressource*) qui joue un rôle ;
- Les *flots de données* décrivent la façon dont les activités s'enchaînent et s'échangent des données.

2) L'éditeur graphique d'APEL

Notre équipe dispose d'un éditeur graphique (Figure 22), permettant de construire des modèles de procédés conformément au méta-modèle d'APEL. Il permet également de contrôler la cohérence du modèle de procédé construit.

Cet éditeur offre un langage graphique très simple et de haut niveau d'abstraction. La Table 6 présente un résumé de ce langage graphique. Nous trouvons une présentation plus détaillée de cet éditeur dans le tutorial [Fra 02].

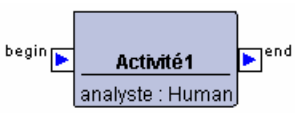
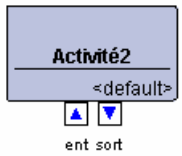
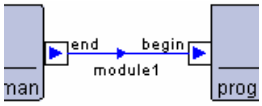
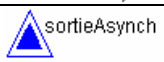
Activité portant le nom "Activité1", contenant un port d'entrée synchrone "begin" et un port de sortie synchrone "end". Le responsable de cette activité est un humain, ayant le rôle d'analyste.	
Activité portant le nom "Activité2", contenant un port d'entrée asynchrone "ent" et un port de sortie asynchrone "sort". Le responsable de cette activité sera défini pendant l'exécution.	
Flot de données entre deux activités. A la fin de la première activité, le produit nommé "module1" est envoyé de la première activité vers la deuxième à travers ce flot de données.	
Port d'entrée synchrone de l'activité englobante, nommée "begin".	
Port de sortie synchrone de l'activité englobante, nommée "end".	
Desktop (Poste de travail), il forme un type particulier de port contenu dans l'activité englobante.	
Flot de données entre le port d'entrée de l'activité englobante, et le Desktop.	
Port d'entrée asynchrone de l'activité englobante, nommée "entreeAsynch".	
Port de sortie asynchrone de l'activité englobante, nommée "sortieAsynch".	

Table 6 : Un résumé du langage graphique de l'éditeur d'APEL

La Figure 22 présente un exemple de modèle de procédé pour la rédaction d'un document réalisé à l'aide de l'éditeur APEL. Un humain jouant le rôle de "Author" sera l'auteur responsable de l'activité "Edit". Il reçoit les instructions afin d'exécuter la tâche "Edit". Il envoie ensuite le document rédigé au rapporteur (humain jouant le rôle de "Reviewer") qui, dans sa tâche "Review", décide si le document est valide (OK) ou si il doit être repris (KO). Dans le cas où le document doit être repris, il est retourné à un auteur (un humain jouant le rôle de "Author", pas nécessairement le même humain qu'au début) avec les corrections à apporter, afin que la tâche "Edit" puisse être de nouveau exécutée. Dans le cas où le document est valide, il est envoyé au Desktop (Poste de travail) du responsable de l'activité englobante. Celui-ci l'envoie manuellement au port de sortie "end" du procédé, le procédé est alors terminé.

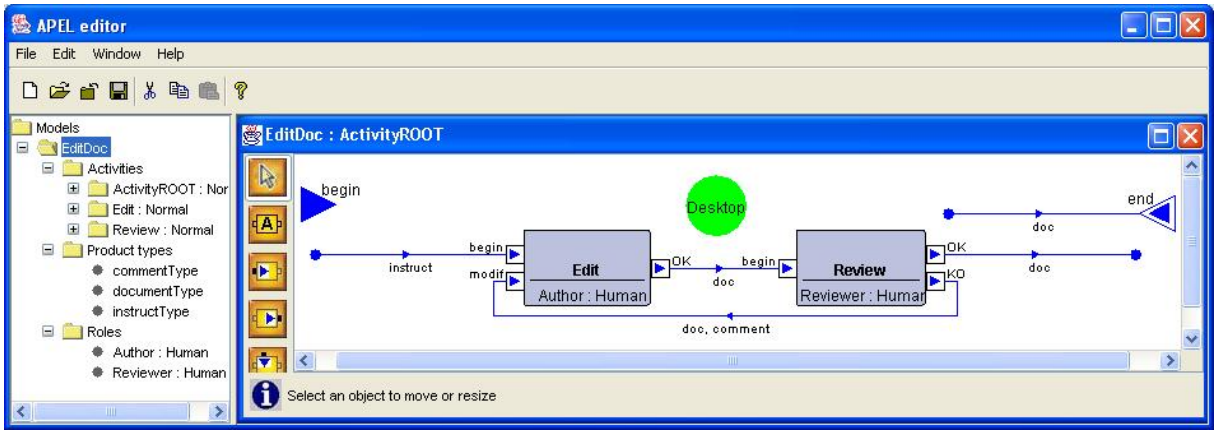


Figure 22 : L'éditeur graphique d'APEL - un exemple de modèle de procédé

2.2. L'Application du Domaine

Après avoir défini le méta-modèle du domaine d'applications, et construit, conformément à ce méta-modèle, la Machine Virtuelle de ce domaine d'applications, la deuxième étape de conception du logiciel à l'aide de la plate-forme Mélusine consiste à construire l'Application du Domaine.

L'architecture des applications conçues à l'aide de Mélusine est composée de trois couches principales : la couche conceptuelle, la couche médiateur, et la couche outils.

2.2.1. La couche conceptuelle

Nous définissons la couche conceptuelle d'un logiciel conçu à l'aide de Mélusine de la manière suivante :

> La *couche conceptuelle* contient le modèle métier décrivant la logique de ce logiciel. Ce modèle est défini conformément au méta-modèle du domaine métier de ce logiciel.

1) Un peu d'histoire…

Au début de nos travaux dans le domaine de Fédération, la couche conceptuelle était appelé l'Univers Commun. Ce dernier était défini comme étant "la réification des relations entre les composants de domaines distincts" [Vil 03].

L'Univers Commun a été utilisé pour définir les concepts communs entre deux (ou plusieurs) applications que nous souhaitons faire coopérer pour atteindre un but commun (Figure 23).

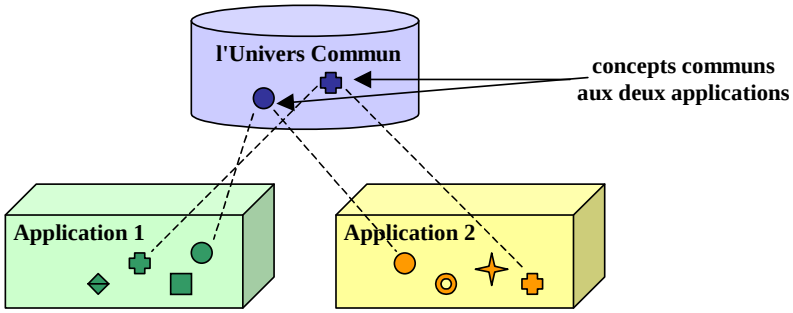


Figure 23 : l'Univers Commun définissant les concepts communs entre deux applications

La coopération entre ces deux applications était assurée à travers ces concepts communs où chacun des concepts d'application se synchronisait avec le concept de l'univers commun correspondant et vice-versa.

Techniquement, cette synchronisation est réalisée se passait à travers un système de notifications : lorsqu'un concept commun d'une application changeait, le concept de l'univers commun correspondant était notifié afin qu'il se synchronise avec le concept modifié. Ce concept d'univers commun notifiait ensuite les autres applications contenant ce même concept commun afin qu'ils se synchronisent à leur tour [Le 04].

2) La couche conceptuelle aujourd'hui

Par la suite, la couche conceptuelle a évoluée en prenant une place plus importante. A présent, cette couche n'est plus utilisée pour contenir uniquement les concepts communs, mais elle est utilisée pour contenir les concepts métier décrivant la logique du logiciel global.

La couche conceptuelle contient donc le modèle métier décrivant la logique du logiciel que nous souhaitons concevoir. Ce modèle est construit conformément au méta-modèle du domaine de ce logiciel que nous avons défini.

A l'exécution, une instance du modèle défini s'exécutera dans la couche conceptuelle du logiciel, à l'aide de la Machine Virtuelle que nous avons construit. Cette instance coordonnera l'exécution des outils participants au logiciel suivant la logique que nous avons spécifiée dans le modèle métier.

3) Exemple : un logiciel dirigé par procédés conçu à l'aide de Mélusine

Nous présentons, tout au long de ce chapitre, un exemple de logiciel de rédaction de document dirigé par procédés, afin d'illustrer la conception de logiciel à l'aide de Mélusine.

Pour concevoir notre logiciel de rédaction de document dirigé par procédés, il nous faut construire le modèle du logiciel, conformément au méta-modèle du domaine de ce logiciel. Ce logiciel de rédaction de document entre dans le domaine de procédés. Le modèle de ce logiciel doit alors être construit à l'aide d'APEL, notre Machine Virtuelle du Domaine de Procédés.

Nous avons illustré l'édition du modèle de notre logiciel de rédaction de document à travers notre éditeur graphique d'APEL dans la Figure 22. Une instance de ce modèle sera exécutée, à l'exécution de notre logiciel, à l'aide du moteur APEL intégré dans Mélusine. Cette instance forme la couche conceptuelle du logiciel, elle coordonnera l'exécution du logiciel suivant la logique que nous avons spécifiée.

La Figure 24 illustre une instance de modèle du logiciel de rédaction de document, au cours de son exécution. Les responsables des activités du procédé sont affectés : Mr. Jones, jouant le rôle d'auteur, est responsable de l'activité Edit ; Mr. Smith jouant le rôle de rapporteur, est responsable de l'activité Review.

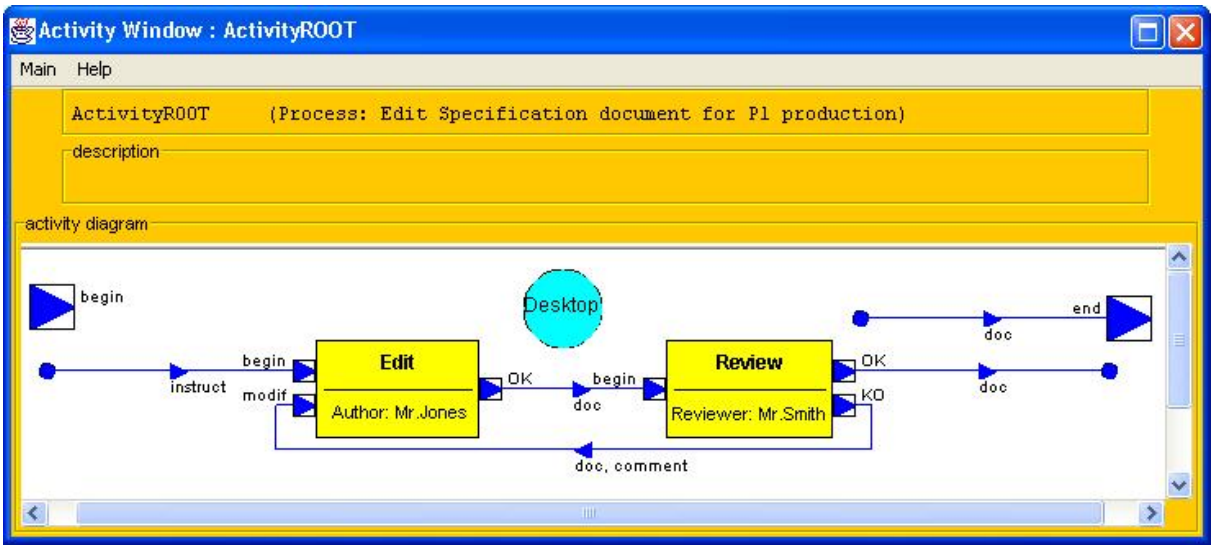


Figure 24 : L'exécution du logiciel de rédaction de document - la couche conceptuelle

Le moteur APEL[50], intégré dans Mélusine, se charge de l'exécution de cette instance de procédé. Il gère les transitions d'états des activités de cette instance conformément au diagramme d'états illustré dans la Figure 25. Nous remarquons, dans ce diagramme d'états, qu'une activité se réinitialise (redevient dans l'état INIT) après sa terminaison lorsque son activité mère est terminée ou interrompue. L'activité, après avoir terminé, peut également redevenir dans l'état READY si un de ses ports d'entrées est de nouveau sollicité. En conséquence, l'activité peut continuer indéfiniment à changer d'état pendant l'exécution du procédé. Ainsi, le diagramme d'états des activités dans le moteur APEL ne contient pas d'état final.

[50] [Vil 02] présente une spécification formelle complète du fonctionnement du moteur APEL.

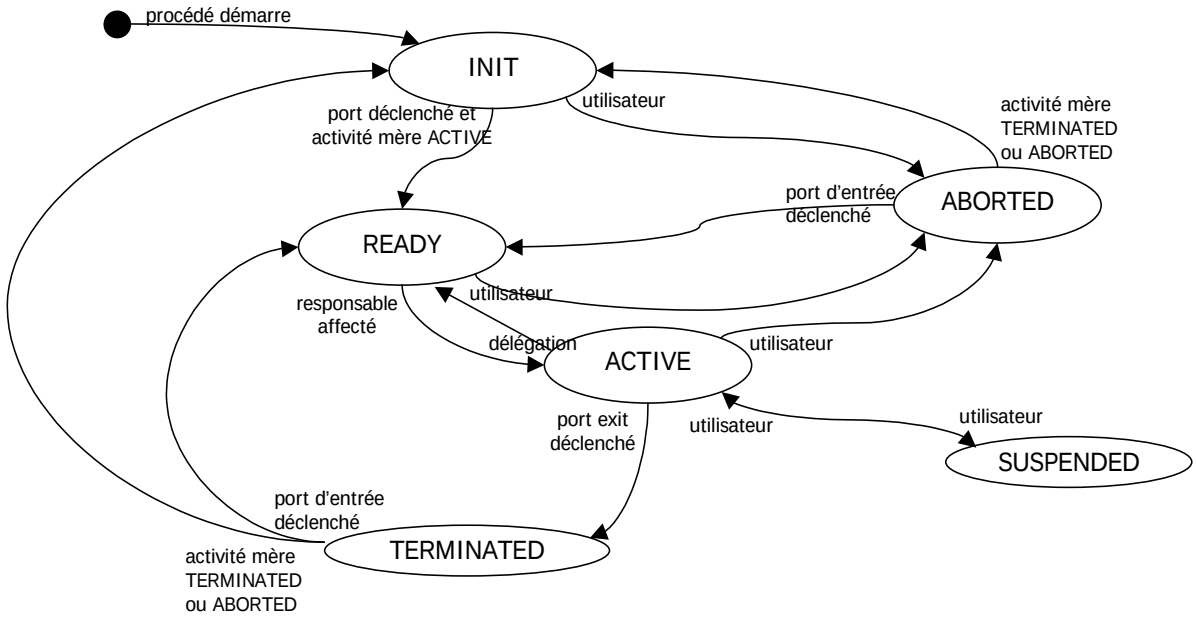


Figure 25 : Le diagramme d'états des activités dans le moteur APEL [Vil 02]

Une activité du procédé peut avoir un des états suivants :

état :	
INIT	L'activité est à l'état initial (démarrage du procédé).
READY	L'activité est prête à recevoir un responsable, afin de démarrer.
ACTIVE	Un responsable a été affecté à l'activité, et celle-ci a démarré.
TERMINATED	L'activité est terminée.
ABORTED	L'activité a été interrompue.
SUSPENDED	L'activité est suspendue.

Le moteur APEL permet l'évolution dynamique, c.-à-d. qu'il est possible à tout instant de modifier le modèle de procédé ou une de ses instances pendant son exécution. Par exemple, nous pouvons ajouter ou supprimer des activités, ajouter ou supprimer des flots de données, ajouter ou supprimer des produits ou des ressources… Les changements effectués sur une instance n'affectent que cette instance, par contre une modification du modèle affecte les futures instances de ce modèle, mais pas les instances en cours d'exécution.

Le moteur APEL a également la bonne propriété de reprise sur pannes. C'est à dire qu'il est capable en cas de panne d'être relancé avec le même état dans lequel il se trouvait avant la panne. Pour cela, une image de l'état du moteur d'APEL est sauvegardée à chaque changement.

2.2.2. La couche médiateur

Nous avons vu que l'architecture des logiciels conçus à l'aide de Mélusine est composée de trois couches : conceptuelle, médiateur, et outils. Nous avons vu également que la couche conceptuelle contient le modèle métier décrivant la logique du logiciel.

La couche conceptuelle se charge de coordonner les outils participants au logiciel, suivant la logique décrite dans le modèle métier. Ces outils permettent de communiquer avec les ressources humaines (via l'outil Agenda par exemple), ou de réaliser certaines tâches (via l'outil Compilateur par exemple).

Les liens entre les concepts du modèle métier se trouvant dans la couche conceptuelle, et ces outils ne doivent pas être codés au niveau conceptuel pour au moins deux raisons : D'une part, car cela fait apparaître, au niveau conceptuel, des détails qui ne sont pas du même niveau que les concepts métier du logiciel, ce qui pollue notre niveau conceptuel. Et d'autre part, car ceci rigidifie la liaison entre le modèle et les outils. Le besoin d'implémenter un concept par un nouvel outil impliquerait alors de modifier le niveau conceptuel.

Ce besoin de substituer un outil par un autre est d'une grande importance aujourd'hui. Surtout avec l'utilisation des services distants et des services web qui impose de nouvelles contraintes sur les logiciels de coordination, tels que la gestion de la disponibilité de ces outils et de la fiabilité de la connexion réseau. Les logiciels de coordination ont, en conséquence, besoin de contrôler dynamiquement les liens entre le modèle métier et les outils qui l'implémentent, afin de pouvoir substituer un outil par un autre en cas de besoin [HG 03]. La substitution d'un outil par un autre n'est pas une opération évidente, surtout quand les outils utilisés n'emploient pas les mêmes protocoles de communication. La collaboration doit alors être gérée en faisant abstraction des moyens de communication, ce qui signifie la séparation entre la couche de coordination et la communication avec les outils utilisés.

Pour répondre à ces besoins, nous avons ajouté une couche appelée médiateur. Cette couche se charge de faire le lien, de façon plus flexible, entre les concepts du modèle métier et leurs implémentations en terme d'outils réalisant ces concepts.

La couche médiateur contient deux sous-couches : les adaptateurs, et les services. Nous allons présenter dans cette section chacune de ces deux sous-couches.

1) Les adaptateurs

Les adaptateurs se chargent de faire le lien entre les concepts du modèle du logiciel, contenu dans la couche conceptuelle, et les services de la couche médiateur décrivant les outils participants (nous présentons les services dans la section suivante).

Ainsi un adaptateur décrit en quoi consiste un concept (ou une instance de concept) du modèle du logiciel en terme d'exécution d'outils. Il décrit aussi l'influence que peut avoir un outil sur les instances de concepts du modèle du logiciel, en terme de modification, d'ajout et

de suppression. Les adaptateurs peuvent alors être définis, dans le logiciel, comme étant “la matérialisation des règles du jeu” [Vil 03].

Dans notre équipe, nous avons défini un langage de médiation, ressemblant à un langage de programmation, qui nous permet de décrire les adaptateurs de manière simple. La construction des adaptateurs se fait à l’aide de l’éditeur graphique de notre plate-forme Mélusine : "FedeEditor" (Figure 26). Ces adaptateurs sont ensuite vérifiés et compilés par notre plate-forme Mélusine, afin qu’ils puissent être exécutés.

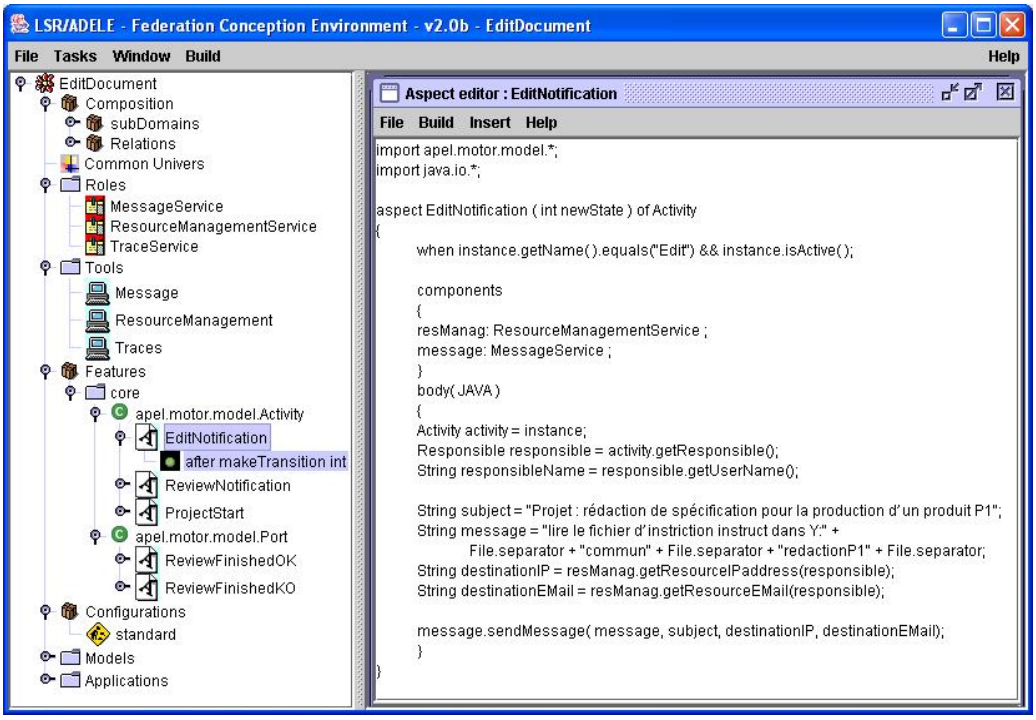


Figure 26 : L’éditeur FedeEditor – l’adaptateur "EditNotification"

Techniquement, un adaptateur est un aspect [AOSD] interceptant une méthode d’une classe de la couche conceptuelle. Pour construire un adaptateur, il faut tout d’abord définir, dans l’éditeur graphique FedeEditor, la méthode interceptée (la méthode "makeTransition" de la classe "apel.motor.model.Activity" dans l’exemple de la Figure 26, cette méthode est sélectionnée dans la partie gauche de la figure), et l’ordre dans lequel l’adaptateur et la méthode sont exécutés (l’adaptateur s’exécute avant la méthode, à la place de la méthode, ou comme dans cet exemple après la méthode). FedeEditor génère ensuite un fichier ".xml" contenant la définition (la méthode interceptée et l’ordre) de chacun de ces adaptateurs.

La Figure 26 illustre un exemple d’adaptateur écrit dans notre langage de médiation et affiché dans notre éditeur FedeEditor. Cet adaptateur, "EditNotification", sera présenté de manière plus détaillée un peu plus tard dans cette section (Code 3). Nous remarquons dans cet exemple que l’adaptateur utilise deux services (nous présentons les services dans la section suivante) : ResourceManagementService et MessageService. Ces services sont utilisés dans le corps de

l'adaptateur, pareillement que des interfaces du langage Java ; nous pouvons effectuer des appels de méthodes sur ces services comme nous le faisons en Java.

Le langage de médiation : un langage extensible

Ce langage que nous avons introduit ci-dessus est un langage générique de la couche médiation. Il fait le lien entre la couche conceptuelle et la couche de services, en transmettant et en adaptant les changements d'une de ces deus couches vers l'autre, à l'aide des instructions générales qu'il contient.

Afin de réaliser une meilleure adaptation entre la couche conceptuelle et la couche de services, il faut spécialiser ce langage pour l'adapter au domaine de l'application du logiciel que nous construisons. Pour cela, nous proposons de l'étendre afin de lui ajouter des instructions correspondants à ce domaine d'application. Ainsi, en étendant notre langage, nous pourrons construire des adaptateurs, manipulant des instances de concepts métier du modèle du logiciel, contenu dans la couche conceptuelle.

Nous avons étendu notre langage de médiation, pour y ajouter des instructions propres au domaine de procédés. Nous présentons dans la section suivante les instructions que nous avons ajoutées.

Le langage de médiation pour le domaine de procédés

Nous avons vu que l'adaptateur a pour fonction de décrire en quoi consiste un concept (ou une instance de concept) du modèle du logiciel en terme d'exécution d'outils. Et de décrire l'influence que peut avoir un outil sur les instances de concepts du modèle du logiciel, en terme de modification, d'ajout et de suppression.

Lorsque nous construisons un logiciel dirigé par procédé, nous avons besoin des adaptateurs pour transmettre l'influence des outils vers les instances du modèle de procédé (les instances d'activités, de produits, de ports...) et vice-versa. Pour cela, il nous faut ajouter au langage de médiation, des instructions spécifiques au domaine de procédés.

J'ai réalisé cette extension du langage de médiation, pour le domaine de procédés. Pour cela, j'ai définis les instructions à ajouter dont nous avons besoin (Table 7). J'ai implémenté chacune de ces méthodes, afin qu'elle réalise l'instruction demandée. Nous remarquons que l'extension de notre langage de médiation pour un domaine métier spécifique forme un DSL (Domain Specific Language [Wil 01]). Seulement le langage de médiation étendu n'a pas le même objectif que les DSL, qui est de définir un langage dans lequel l'application sera "programmée", puis compilée vers du code exécutable [EIV 05].

J'ai ensuite rassemblé ces méthodes dans un "Outil d'Extension du Langage de Médiation". Et j'ai intégré cet outil lors de la construction de logiciel dirigé par procédés, à l'aide de notre plate-forme Mélusine. Ainsi, j'ai pu utiliser ces instructions spécifiques au domaine de

procédés, pour construire mes adaptateurs, afin qu'ils produisent une meilleure adaptation entre la couche de services et la couche conceptuelle contenant le modèle de procédé.

Ajoute une instance de produit nommé `productName` contenant les attributs `attributes` dans le port nommé `portName` de l'activité `activity`	`addProductInstance(activity, portName, productName, attributes);`
Créer un nouveau nom d'instance, du procédé nommé `processName`	`getNewProcessInstanceName (processName);`
Récupère les attributs du produit nommé `productName` contenu dans le port nommé `portName` de l'activité `activity`	`getProductAttributes(activity, portName, productName);`
Retourne l'activité nommée `activityName` de l'instance de procédé `processInstance`	`getSubActivity(processInstance, activityName);`
Se charge de lancer le procédé nommé `processName`	`processLaunch(processName);`
Retourne la valeur de l'attribut nommé `attribute` du produit nommé `productName` du Desktop de l'activité `activity`	`getProductAttributeValue(activity, productName, attribute);`
Retourne l'instance du produit nommé `productName` du Desktop de l'activité `activity`	`getProductInstance(activity, productName);`
Assigne la valeur `value` à l'attribut nommé `attribute` du produit nommé `productName` du Desktop de l'activité `activity`	`setProductAttributeValue(activity, productName, attribute, value);`
Termine l'activité `activity` en envoyant tous les produits de son Desktop vers son port de sortie nommé `portName`	`terminateActivity(activity, portName);`

Table 7 : les instructions ajoutées au langage de médiation pour le domaine des procédés

Nous présentons des exemples de l'utilisation de cette extension du langage pour le domaine de procédés lors de la description des adaptateurs de notre application Planification de Voyages présentée à la section 4 de ce chapitre.

2) Les services

Un service est l'abstraction d'un outil offrant un certain nombre de fonctionnalités. Il peut être vu comme la description d'un outil requis par le logiciel.

Chaque concept du modèle dans la couche conceptuelle peut appeler un adaptateur à un moment donné. L'adaptateur contient la séquence d'appels de fonctionnalités des services qui sont censés implémenter le concept.

Les contrats de coordination s'appuient donc sur les services, et non pas directement sur les outils ce qui rend l'évolution du logiciel global plus facile. On peut donc facilement remplacer un outil par un autre, du moment ou il implémente le même service, et ceci sans modifier le niveau conceptuel ni le niveau médiateur.

3) La couche médiateur dans notre logiciel de rédaction de document

Dans notre exemple de logiciel de rédaction de document, nous avons spécifié la couche conceptuelle, en décrivant le procédé de coordination. Ce procédé décrit la logique du logiciel, il se charge de coordonner son exécution.

Pour compléter la réalisation de ce logiciel, nous allons introduire des outils participant à l'exécution. Le lien entre ces outils et le procédé de coordination de la couche conceptuelle se fait à travers la couche de médiateur. Il nous faut donc réaliser les adaptateurs et les services qui se chargeront de relier la couche conceptuelle à la couche des outils.

Nous avons choisi de présenter un exemple simple, intitulé "Rédaction de document", pour faciliter la compréhension de l'architecture à trois couches. Nous présenterons par la suite des exemples plus complexes et plus complet. Nous présentons dans la section suivante (4) la description détaillée du scénario de qu'effectue notre exemple simplifié.

4) Description du scénario de rédaction de document

Pour simplifier notre exemple de rédaction de document, nous supposons que le document rédigé sera enregistré sous le nom de "doc", dans un répertoire du serveur commun, dont l'adresse est connue par le logiciel. De la même manière les instructions de rédaction rédigés par le chef de projet et les commentaires du rapporteur seront enregistrés dans le même répertoire partagé du serveur sous le nom de "instruct" et "comment".

Au moment de l'exécution, le déroulement de notre logiciel de rédaction simplifié est le suivant :

- Le chef du projet démarre, à l'aide du moteur APEL, une instance de procédé de rédaction de document, afin de rédiger un document de spécification pour la production d'un produit P1 ;
- Il rédige les instructions, et les enregistre sous le nom de "instruct", dans un répertoire partagé du serveur commun : "Y :\commun\redactionP1\" ;
- Il utilise un outil "Gestionnaire de ressources", qui lui affiche les ressources disponibles pour chacun des deux rôles auteur et rapporteur afin de choisir la personne responsable de chacune des deux activités de l'instance de procédé ;
- Lorsque le choix des responsables est fait, l'activité Edit devient active. Le responsable de l'activité Edit est alors notifié à l'aide d'un outil de "Messagerie". L'outil de messagerie a pour rôle d'informer le responsable de chaque activité du

travail qui lui est demandé. Il informe donc le responsable de l'activité Edit qu'il doit lire le fichier "instruct" dans le répertoire commun ;

- Lorsque le responsable de l'activité Edit a fini son travail de rédaction, il enregistre la spécification demandée sous le nom de "doc" dans le répertoire commun. Il termine ensuite l'activité à l'aide de l'outil de supervision d'APEL intégré dans Mélusine ;
- L'activité Edit est alors terminée et c'est l'activité Review qui devient active. Le responsable de l'activité Review est alors notifié (toujours à l'aide de l'outil de messagerie) qui l'informe de rapporter le fichier "doc" dans le répertoire commun ;
- Lorsque ce travail de relecture est terminé, le responsable de l'activité Review termine à son tour l'activité soit en considérant que le document est bon (port OK), soit en considérant que ce document doit être modifié (port KO). Dans ce cas, il enregistre ses commentaires sous le nom de "comment" dans le répertoire commun ;
- Si le rapporteur a validé le document (OK), le chef du projet est notifié (à l'aide de l'outil de messagerie) que l'instance de procédé a terminé son exécution ;
- Si le rapporteur n'a pas validé le document (KO), l'instance de procédé va revenir à l'activité Edit. Au même moment, le chef du projet est également notifié pour choisir (à l'aide de l'outil "Gestionnaire de ressources") un nouvel auteur et un nouveau rapporteur comme responsables des deux activités. L'exécution de l'instance de procédé reprend à partir du quatrième point de cette liste.

Pour réaliser cet exemple, nous avons besoin de deux outils :

- Un outil de gestion de ressources pour afficher la liste des ressources disponibles correspondantes à un rôle spécifique et pour permettre de choisir dans cette liste un responsable pour chaque activité ;
- Un outil de messagerie pour notifier le responsable de l'activité du travail qui lui est demandé.

Nous créons donc un service de messagerie (Code 1) et un service de gestion de ressources (Code 2).

```
public interface MessageService {
   public void sendMessage( String message, String subject, String destinationIP,
                                  String destinationEMail ); }
```

Code 1 : Le service de messagerie

```
import java.util.Set;
public interface ResourceManagementService {
     public Set getAvailableResources( String role );
     public void setResponsible( String resource, String activity );
     public String getResourceEMail( String resource );
     public String getResourceIPaddress( String resource ); }
```

Code 2 : Le service de gestion de ressources

Pour lier les concepts de la couche conceptuelle à ces deux services, nous construisons des adaptateurs correspondants. Ces adaptateurs doivent décrire en quoi consistent les concepts ou les instances de concepts du modèle de procédé en terme d'appels de fonctionnalités des services. Nous allons présenter ici un exemple de ces adaptateurs.

Nous avons vu dans la description du déroulement de notre logiciel simplifié de rédaction de documents que lorsque le choix des responsables est effectué, l'activité Edit devient active. Le responsable de l'activité Edit est alors notifié à l'aide d'un outil de "Messagerie" qui l'informe qu'il doit lire le fichier "instruct" dans le répertoire commun. Pour construire cet adaptateur de notification du responsable de l'activité Edit, nous utilisons notre langage de médiation (Code 3). Cet adaptateur se charge de vérifier que l'instance interceptée est une instance de l'activité "Edit" et qu'elle est dans l'état actif. Ceci correspond aux conditions demandées pour la notification.

Nous déclarons dans la partie "components" les services utilisés dans cet adaptateur. Dans le corps de l'adaptateur, nous retrouvons, à partir de l'instance de l'activité interceptée, le nom de son responsable. Les deux services "ResourceManagementService" et "MessageService" sont ensuite utilisés. Le premier pour retrouver, à partir du nom du responsable, son adresse électronique et l'adresse IP de sa machine. Le deuxième pour envoyer un message à ce responsable à l'aide des informations obtenues.

```
import apel.motor.model.*;
import java.io.*;
aspect EditNotification ( int newState ) of Activity{
  when instance.getName( ).equals("Edit") && instance.isActive( );
  components {
    resManag : ResourceManagementService ;
    message : MessageService ;
  }
  body( JAVA ) {
    Activity activity = instance;
    Responsible responsible = activity.getResponsible();
    String responsibleName = responsible.getUserName();
    String subject = "Projet : rédaction de spécification pour la production d'un produit P1";
    String message = "lire le fichier d'instructions instruct dans Y :" + File.separator +
                     "commun" + File.separator + "redactionP1" + File.separator;
    String destinationIP = resManag.getResourceIPaddress(responsibleName);
    String destinationEMail = resManag.getResourceEMail(responsibleName);
    message.sendMessage( message, subject, destinationIP, destinationEMail);
  }
}
```

Code 3 : L'adaptateur de notification du responsable de l'activité Edit : "EditNotification"

La Figure 27 présente l'architecture à trois couches de notre logiciel simplifié de rédaction de document. Elle illustre aussi l'adaptateur de notification du responsable de l'activité Edit que nous venons de détailler (Code 3).

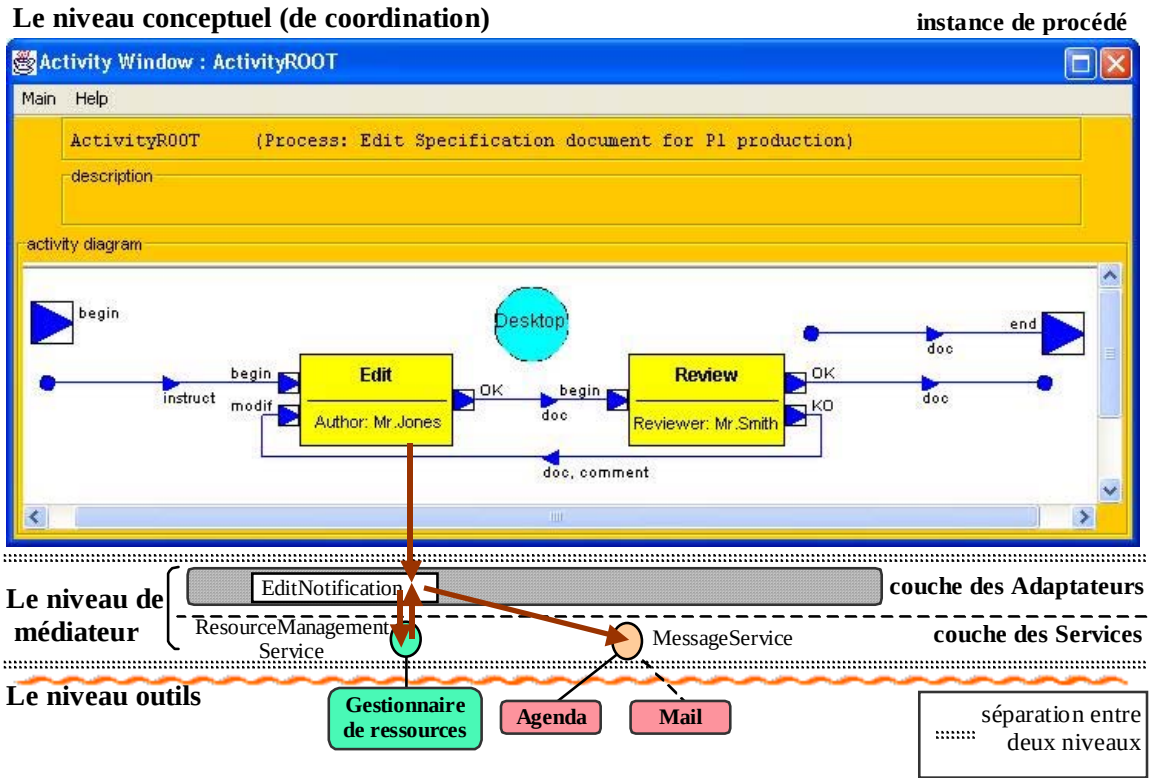


Figure 27 : L'architecture à trois couches du logiciel de rédaction de document

Pour compléter le développement de notre logiciel de rédaction de document, il nous faut construire les adaptateurs suivants :

- un adaptateur pour notifier le responsable de l'activité Review ;
- un adaptateur se déclenchant lorsque chef du projet démarre une instance de procédé. Sa fonction sera de lancer l'outil "Gestionnaire de ressources" pour permettre au chef du projet de choisir le responsable de chacune des deux activités et de transmettre ce choix vers les activités de l'instance de procédé ;
- un adaptateur, semblable au précédant, se déclenchant lorsque le rapporteur choisit de terminer son activité Review par le port KO. A ce moment-là, l'outil "Gestionnaire de ressources" doit être de nouveau lancé afin que le chef du projet choisisse une nouvelle fois les responsables des activités ;
- et un adaptateur, utilisant le service de messagerie, pour notifier le chef du projet que l'instance de procédé a terminé son exécution, dans le cas où le rapporteur choisit de terminer son activité Review par le port OK.

Nous remarquons dans la Figure 27 que le service de messagerie est lié à deux outils : "Agenda" et "Mail". Ceci car ces deux outils implémentent le service de messagerie, il est

donc possible de remplacer l'un de ces deux outils par l'autre, sans avoir à modifier quoi que ce soit dans notre logiciel.
Le choix de l'outil utilisé est fait avant le démarrage du logiciel, en sélectionnant l'outil que l'on préfère. Ceci ce fait dans l'éditeur graphique de notre plate-forme Mélusine : "FedeEditor", dont nous avons déjà parlé précédemment.

2.2.3. La couche outils

Mélusine nous permet de concevoir et d'exécuter des systèmes coordonnant plusieurs types d'outils. Ce qui la différencie des autres systèmes, comme les systèmes d'orchestration, se limitant pour la plupart d'entre eux à la coordination de services web.

Actuellement, Mélusine permet d'intégrer trois catégories d'outils :

- Les outils distants : lorsque l'outil est distant, cette liaison entre l'outil et le service est faite à travers un "proxy" et un "wrapper" qui se chargent d'adapter les fonctionnalités offertes par l'outil aux fonctionnalités demandées par le service (outil "Mail" dans la Figure 28). La communication entre le proxy et le wrapper est géré par la plate-forme Mélusine. Ainsi, le concepteur du logiciel n'a pas à gérer cette communication ;

- Les services web : dans le cas des services web, la liaison se fait directement entre le proxy et le service web, dans ce cas-là pas besoin de wrapper (l'outil "Agenda" dans la Figure 28). La communication entre le proxy et le service web est également géré par la plate-forme Mélusine, le concepteur du logiciel n'a pas à gérer cette communication ;

- Les outils locaux : quand l'outil est local, et fait sur mesure, il n'a pas besoin d'intermédiaire pour adapter ses fonctionnalités à celles demandées par le service. Cet outil est alors directement lié au service, c'est le cas de l'outil "Gestionnaire de ressources" dans la Figure 28.

Un service définit de manière abstraite les fonctionnalités d'une catégorie d'outils. Il permet de regrouper les outils similaires et en fonction du contexte d'exécution, il est possible de choisir l'exécutant le plus adéquat. Cela permet de substituer un outil implémentant ce service par une autre sans pour cela modifier la couche conceptuelle ni la couche médiateur. Il suffit de préciser à l'exécution lequel des outils implémentera le service.

L'utilisation des adaptateurs pour appeler les fonctionnalités des services implémentant l'activité permet également une grande flexibilité, du fait qu'on peut employer le service dans plusieurs adaptateurs, et qu'on peut modifier l'adaptateur facilement pour avoir une exécution différente.

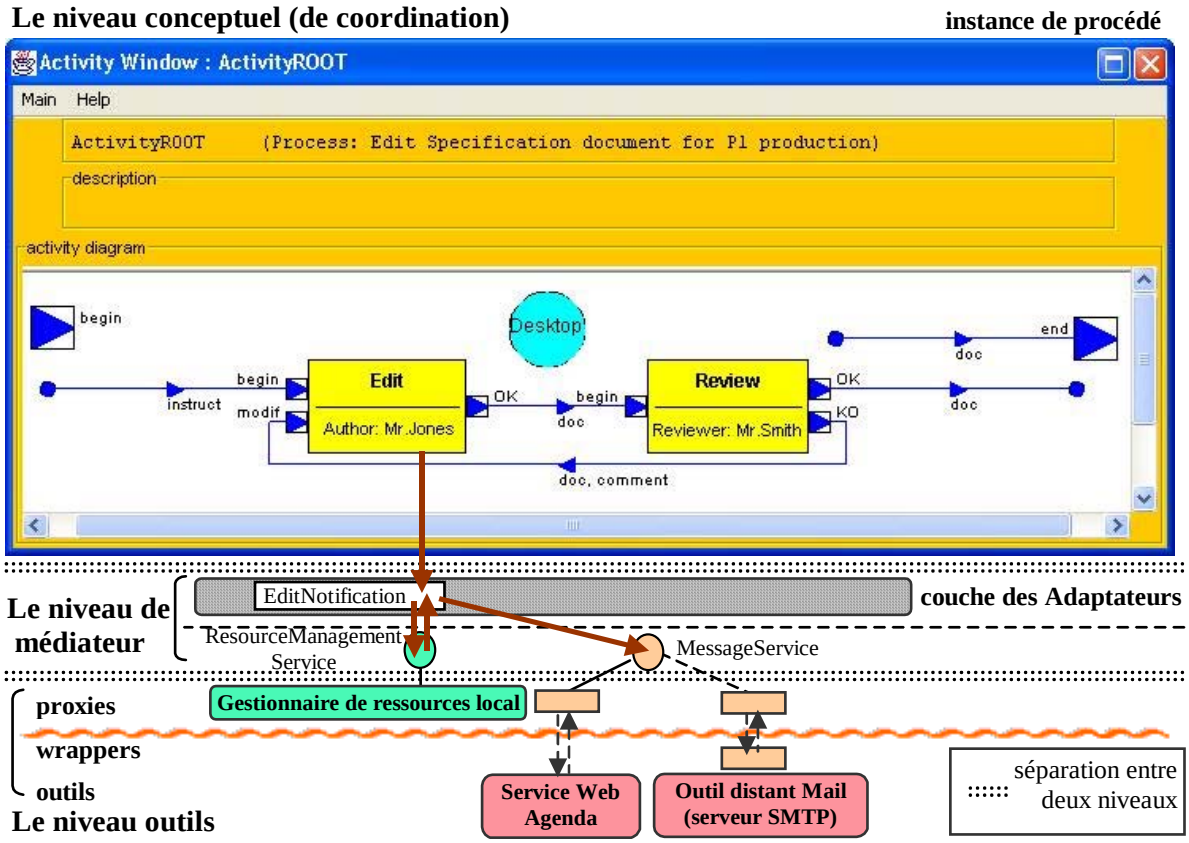


Figure 28 : L'architecture à trois couches des logiciels conçus à l'aide de Melusine

3. La Démo Métier du projet Centr'Actoll : une implémentation industrielle construite à l'aide de Mélusine

Pour faciliter la compréhension, nous avons illustré la présentation de notre plate-forme Mélusine par des exemples simples. Cependant, Mélusine a été utilisée dans des cas industriels réels. Nous présentons dans cette section un de ces cas industriels : la Démo Métier du projet Centr'Actoll qui est une implémentation que j'ai réalisée.

3.1. Le projet Centr'Actoll

Dans le cadre de mon travail sur la Fédération, j'ai participé au projet industriel Centr'Actoll [Centr'Actoll], qui est une collaboration entre plusieurs sociétés :

- ACTOLL (SSII spécialisée dans les métiers du transport et de la monétique) ;
- AREA (péage autoroutes Rhône-Alpes) ;
- INOVATEL (micro nano technologies) ;
- ASK (cartes à puce) ;
- et notre laboratoire LSR (Logiciels, Systèmes et Réseaux).

Ce projet, financé en partie par le Ministère de l'Industrie, vise créer des standards et développer des briques réutilisables dans les domaines de la billettique et du péage.

Compte tenu de la taille du projet, celui-ci a été découpé en sous-projets. Ma participation entre dans le cadre d'un des sous-projet de Centr'Actoll nommé : Site e-commerce AREA.

3.2. Le sous-projet "Site e-commerce AREA"

Dans le cadre du sous-projet Site e-commerce AREA, j'ai été responsable de la réalisation d'une Démo Métier. L'objectif de ce sous projet était d'enrichir le site web d'AREA par de nouvelles fonctionnalités nécessitant de pouvoir d'une part communiquer entre le site web et les serveurs centraux d'AREA et d'autre part de faire évoluer la liaison existante entre les points de vente et les serveurs centraux d'AREA. Ces évolutions devaient se faire sans perturber le système de production et en maintenant la configuration pré-existante. Les partenaires de ce sous-projet étaient ACTOLL, AREA, et le LSR.

3.3. La description de la Démo Métier

La société AREA possède, dans ses locaux à Bron, des serveurs centraux contenant le logiciel de gestion des abonnements pour le passage au péage des autoroutes en Rhône-Alpes ainsi que la base de données (la BD LUCIE) contenant toutes les données sur les abonnés et leur type d'abonnement (numéro 1, Figure 29).

AREA possède également des "Points de Vente" répartis sur la région Rhône-Alpes. Ces points de vente permettent de souscrire à un type d'abonnement choisi (numéro 2, Figure 29).

Lorsqu'une personne souhaite s'abonner, elle doit se présenter dans un point de vente d'AREA afin de donner les informations demandées pour l'inscription (nom, adresse, type d'abonnement...), payer les frais d'abonnement et récupérer le télébadge qui sera collé sur le pare-brise de la voiture. Les informations enregistrées sur le terminal du point de vente sont alors transmises vers la BD LUCIE du serveur central à Bron. De la même manière, pour une modification de ses données, l'abonné doit se présenter dans un point de vente d'AREA afin que ses demandes de modification soient prisent en compte par la base de données.

Afin d'améliorer son service d'abonnement, la société AREA a souhaité simplifier ses procédures d'inscription en offrant à ses clients la possibilité de se pré-inscrire par Internet, de consulter son compte et de pouvoir le modifier en utilisant le site web d'AREA.

La pré-inscription consiste à enregistrer dans une base de données temporaire toutes les informations concernant la personne souhaitant s'abonner. Ainsi, lorsque cette personne se présente au point de vente pour payer et récupérer le télébadge, il suffit de confirmer son inscription pour transmettre les données vers la BD LUCIE sur le serveur central d'AREA. Le temps de saisie des données sera alors économisé.

La consultation du compte consiste à chercher les données concernant l'abonnement dans la BD LUCIE. S'ils ne sont pas trouvés dans cette BD, il faut les chercher dans la BD temporaire de pré-inscriptions. Une fois les données récupérées, il suffit de les afficher sur le site web.

L'utilisateur peut, s'il le souhaite, modifier ces informations. La modification se fait alors en deux temps. La première étape consiste à enregistrer les modifications dans la base de données temporaire. La deuxième étape permet de vérifier les nouvelles données par un employé d'AREA (numéro 5, Figure 29). Si les modifications sont acceptées, elle sont alors intégrées dans la BD LUCIE. Par contre, si elles sont refusées elles sont effacées de la BD temporaire.

Pour réaliser tout cela, il fallait enrichir le site web d'AREA par ces nouvelles fonctionnalités : pré-inscription, consultation du compte de l'abonné, et modification de ce compte. Et créer la liaison entre le serveur hébergeant ce site web, et le serveur central d'AREA, ainsi que les points de vente, et ceci sans perturber le système[51].

Le sous-projet "Site e-commerce AREA" consistait à étendre le système de gestion des abonnements existants en ajoutant ces fonctionnalités. La Démo Métier que nous avons réalisée était une simulation de cette extension du système de gestion des abonnements.

3.4. La réalisation de la Démo Métier

Pour réaliser cette Démo Métier, nous avons utilisé un clone du site web AREA, hébergé chez ACTOLL (numéro 4, Figure 29), ainsi qu'un clone de Point de Vente AREA (numéro 2, Figure 29), et un clone du serveur AREA-SERVEUR contenant une base de données complètement conforme à la réalité (identique à la vraie base de données BD LUCIE, seulement elle n'est pas mise en exploitation). Cette base de données inclut toutes les informations sur les abonnements (numéro 1, Figure 29).

Nous avons installé sur une quatrième machine (numéro 3, Figure 29) notre plate-forme Mélusine afin de prendre en charge l'exécution de l'extension que nous avons construit. Nous avons choisi d'exécuter cette extension sur une quatrième machine pour simuler l'exécution à distance.

[51] Le système de gestion des abonnements, s'exécutant sur le serveur d'AREA et les points de vente, est écrit en langage C++, et s'exécute sur le système d'exploitation Unix.

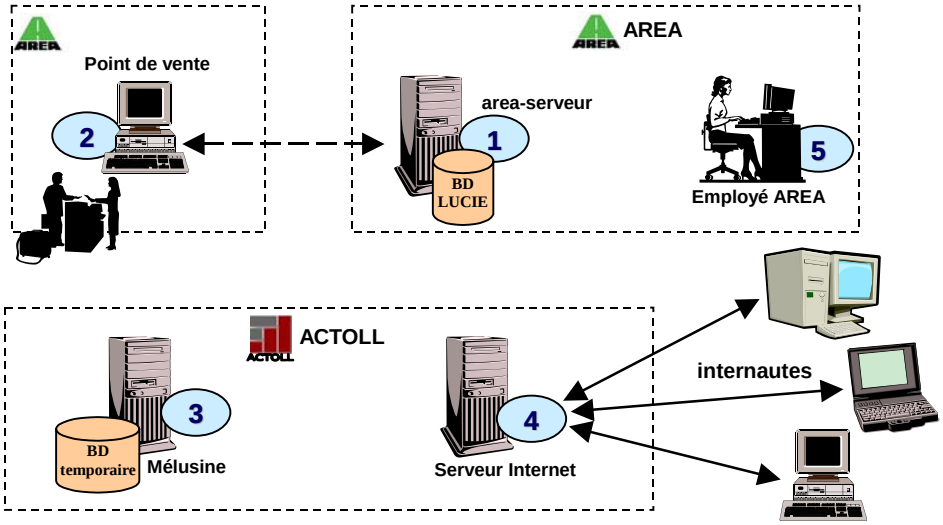


Figure 29 : Démo métier du sous-projet de Centr'Actoll : "Site e-commerce AREA"

Nous avons utilisé Mélusine pour construire notre logiciel d'extension. Nous avons donc défini le méta-modèle métier de notre logiciel (Figure 30). Ce méta-modèle contient trois classes : Abonné, Identifiant et Modif, représentant les trois modèles de données métier utilisées dans notre Démo Métier. Nous avons également construit la Machine Virtuelle du domaine Area, qui est le moteur gérant les modèles de données métier de notre application. Dans cette Demo Métier ce moteur est simple, il contient une seule classe : AreaWebDomain.

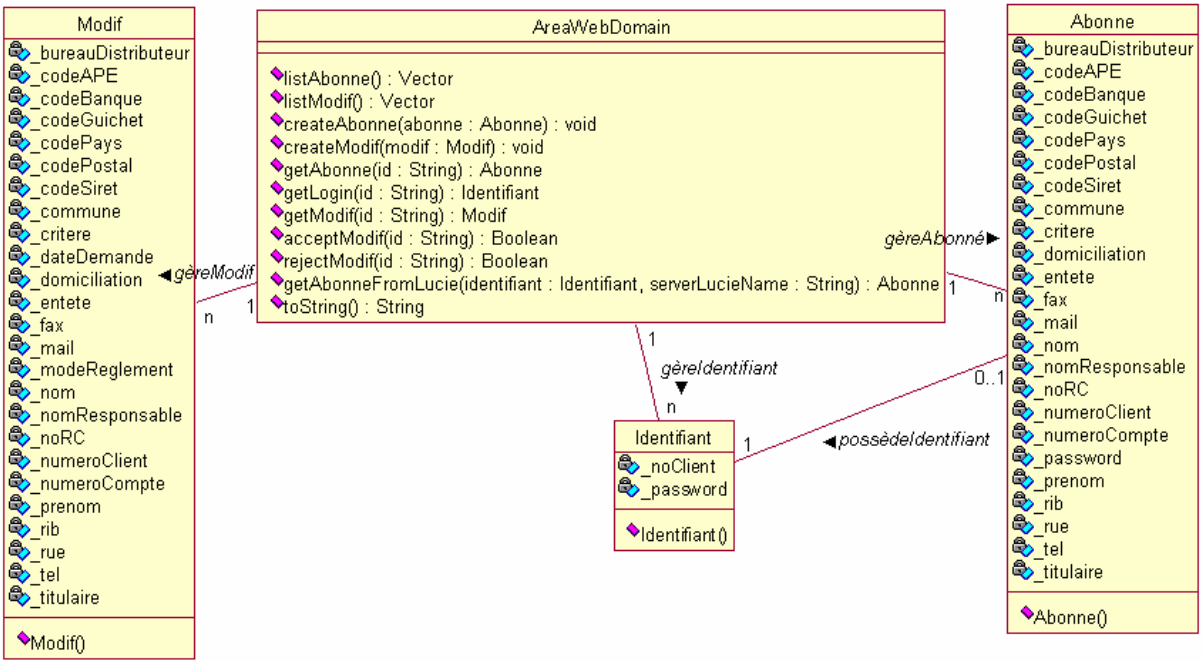


Figure 30 : La couche conceptuelle de notre logiciel d'extension pour la Démo métier

Nous avons ensuite construit quatre outils :

1) L'outil AreaWebSiteTool

L'outil AreaWebSiteTool est un outil distant hébergé sur la machine contenant le serveur du site web (numéro 4, Figure 29). Cet outil se charge de transmettre à la couche conceptuelle, à travers des adaptateurs, les demandes de pré-inscription, de consultation de compte d'abonné et de modification de compte. Lors d'une demande de consultation de compte, cet outil récupère le résultat de la requête de la couche conceptuelle et le transmet au serveur du site web pour l'afficher à l'utilisateur ;

2) L'outil LocalBDManagerTool

L'outil LocalBDManagerTool est un outil local hébergé sur la machine contenant notre logiciel construit sur la plate-forme Mélusine (numéro 3, Figure 29). Cet outil se charge de gérer la base de données temporaire contenue sur cette machine. Nous enregistrons dans cette base de données temporaire les pré-inscriptions d'abonnements et les demandes de modification effectuées à travers le site web. Cet outil se charge (à travers des adaptateurs) de l'enregistrement, de la sélection ou de la suppression des pré-inscriptions et des demandes de modification dans la base de données temporaire lorsque la couche conceptuelle le lui demande ;

3) L'outil ConfirmationTool

L'outil ConfirmationTool est un outil distant hébergé sur la machine de l'employé d'AREA. Il se charge de vérifier chacune des demandes de modifications enregistrées dans la BD temporaire pour décider si elle est acceptée ou refusée (numéro 5, Figure 29). Pour cela, l'outil sollicite la liste des demandes de modifications de la couche conceptuelle et lui renvoie l'acceptation ou le refus de chacune de ces modifications afin que celle-ci transmette au bon outil l'instruction d'appliquer la modification au compte de l'abonné enregistré dans la BD LUCIE, ou l'instruction d'effacer la demande de modification de la BD temporaire ;

4) L'outil LucieTool

L'outil LucieTool est un outil distant qui peut être hébergé sur la machine de notre choix. Dans notre Démo Métier, nous avons choisi de l'héberger sur la machine contenant la plate-forme Mélusine (numéro 3, Figure 29), pareillement à un outil local. Mais l'outil est conçu de façon à pouvoir s'exécuter à distance, en communiquant à travers le réseau avec la couche conceptuelle.

Nous avons construit cet outil de façon à ce qu'il forme un tunnel, à travers lequel passent les communications entre le logiciel de gestion des abonnements (machine numéro 1, Figure 29), et les Points de Vente (PDV, numéro 2, Figure 29). Ainsi, les requêtes envoyées entre ces deux logiciels passent à travers l'outil LucieTool, qui peut éventuellement modifier le message envoyé. La modification est utilisée lors de la confirmation d'une pré-inscription d'un client, car à ce moment-là, la requête de recherche du client envoyée vers BD LUCIE retournera un résultat "null", et c'est à la réception de ce résultat que notre outil le remplace par les détails de la pré-inscription du client qu'il a trouvé dans la BD temporaire. Ces détails

s'afficheront alors sur l'écrans de l'employé du PDV. Celui-ci n'aura plus qu'à compléter ces informations et enregistrer l'inscription du client dans la BD LUCIE.

Ce tunnel offre également un point d'entrée vers le logiciel de gestion des abonnements. Ainsi LucieTool forme la liaison entre la couche conceptuelle et les logiciels d'AREA, et ceci sans modifier ces logiciels.

3.5. Conclusion sur la Démo Métier

Nous avons réalisé une extension du logiciel de gestion des abonnements d'AREA afin de l'enrichir par de nouvelles fonctionnalités. Nous avons ainsi ajouté à ce logiciel la possibilité pour un client de se pré-inscrire par Internet, de consulter son compte ou de le modifier. Et ceci, sans modifier le logiciel d'AREA.

Cette démo métier à permis de démontrer que nous sommes capables d'étendre de vrais logiciels patrimoniaux (legacy), malgré qu'ils soient anciens, plus maintenus, et construits sur des vielles technologies qui ne sont plus utilisées. Et cela afin de les enrichir par de nouvelles fonctionnalités, réalisées en utilisant des technologies très récentes.

Nous avons construit cette extension suivant notre architecture à trois couches s'exécutant au-dessus de notre plate-forme Mélusine. Nous avons ainsi séparé la couche conceptuelle contenant les concepts et la logique de notre extension, des outils implémentant cette extension. Ce qui nous permet de profiter des avantages que nous offre cette architecture, comme la flexibilité de modification de la logique de notre extension et la facilité de substitution d'un outil par un autre.

Nos idées sur l'extension, à ce moment-là, n'étaient pas encore claires. La liaison entre le logiciel d'extension que nous avons réalisé, et le logiciel d'AREA, a été faite à l'aide d'un des outils que nous avons construit. Néanmoins, ce projet a permit de nous indiquer l'importance de l'extension d'applications. Nous nous sommes alors intéressés à cette idée d'extension, et nous avons orienté notre travail de recherche sur cet axe-là.

L'extension a pris, par la suite, une forme différente de ce que nous avons présenté dans la Démo Métier. Elle concerne plus particulièrement la couche conceptuelle, car nous parlons plutôt d'extension de concepts du logiciel que d'extension d'implémentation.

Nous présentons nos travaux sur l'extension d'applications dans le chapitre suivant.

4. Orchestration de services web : une implémentation construite à l'aide de Mélusine

Dans ce travail de recherche, nous nous sommes intéressés plus particulièrement aux procédés, et à la construction de logiciels dirigés par procédés. Nous avons étudié la

coordination d'applications dirigée par procédés, et notamment le domaine de l'orchestration et la chorégraphie de services web présenté dans le chapitre 3.

Nous avons utilisé notre plate-forme Mélusine pour construire des logiciels dirigés par procédés. Nous avons présenté, dans ce chapitre, notre plate-forme Mélusine. Et nous avons illustré notre description de la conception de logiciel à l'aide de cette plate-forme par un exemple de logiciel simplifié de rédaction de document dirigé par procédés, ainsi que par une implémentation industrielle, la Démo Métier du projet Centr'Actoll, que nous avons réalisé à l'aide de notre plate-forme Mélusine.

Nous avons également implémenté une application d'orchestration de services web, en utilisant la plate-forme Mélusine. Pour définir cette application, nous nous sommes inspirés d'un exemple contenu dans la spécification du langage WSCI [AAF 02]. Nous présentons dans cette section cette application de planification de voyages.

4.1. Planification de Voyages : description de l'application

L'application Planification de Voyages a comme objectif de coordonner trois outils s'exécutant à distance : un service web Agence de Voyage, un service web Compagnie Aérienne et l'IHM utilisateur permettant de communiquer avec le voyageur.

Cette coordination vise à réserver des billets d'avion pour effectuer un voyage. Pour cela, le voyageur précise un certain nombre d'informations : la destination, ses préférences de dates, son nom, son adresse, le nombre de personnes… Ces informations sont transmises à l'agence de voyage qui choisit une compagnie aérienne offrant des vols vers la destination demandée. Cette compagnie aérienne est contactée afin de lui demander les disponibilités de vols vers cette destination. La liste des vols disponibles retournée par la compagnie aérienne est envoyée à l'agence de voyage qui choisit un itinéraire. Cet itinéraire est proposé au voyageur. Le voyageur a le choix de réserver les tickets pour cet itinéraire proposé, de demander de changer l'itinéraire, ou d'annuler la planification de voyage.

Si le voyageur choisit d'annuler, l'agence de voyage est notifiée et l'application se termine. S'il choisit de changer l'itinéraire, la demande est transmise à l'agence de voyage qui lui propose un autre itinéraire. Et s'il choisit de réserver les tickets de l'itinéraire proposé, la demande de réservation est transmise à l'agence de voyage en premier et ensuite à la compagnie aérienne afin d'effectuer la réservation. La Figure 31 illustre le modèle de procédé décrivant la logique de notre application Planification de Voyages.

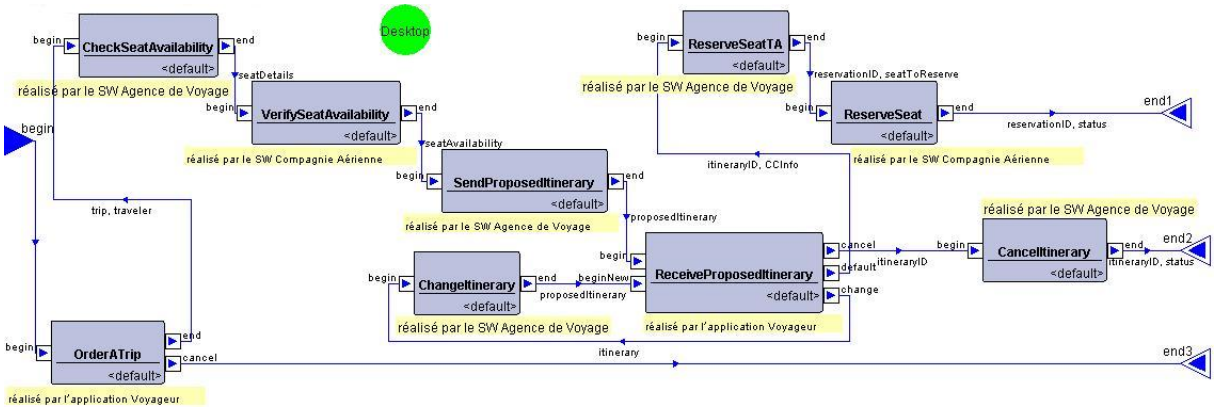


Figure 31 : Le modèle de procédé décrivant la logique de l'application Planification de Voyages

4.2. Planification de Voyages : réalisation de l'application

La couche conceptuelle contient un modèle de procédé que nous avons construit à l'aide d'APEL (Figure 31). Ce modèle de procédé décrit la logique de l'application. A l'exécution de notre application, une instance de ce modèle sera exécutée. Elle coordonnera l'exécution des trois outils : Agence de Voyage, Compagnie Aérienne, et IHM Voyageur.

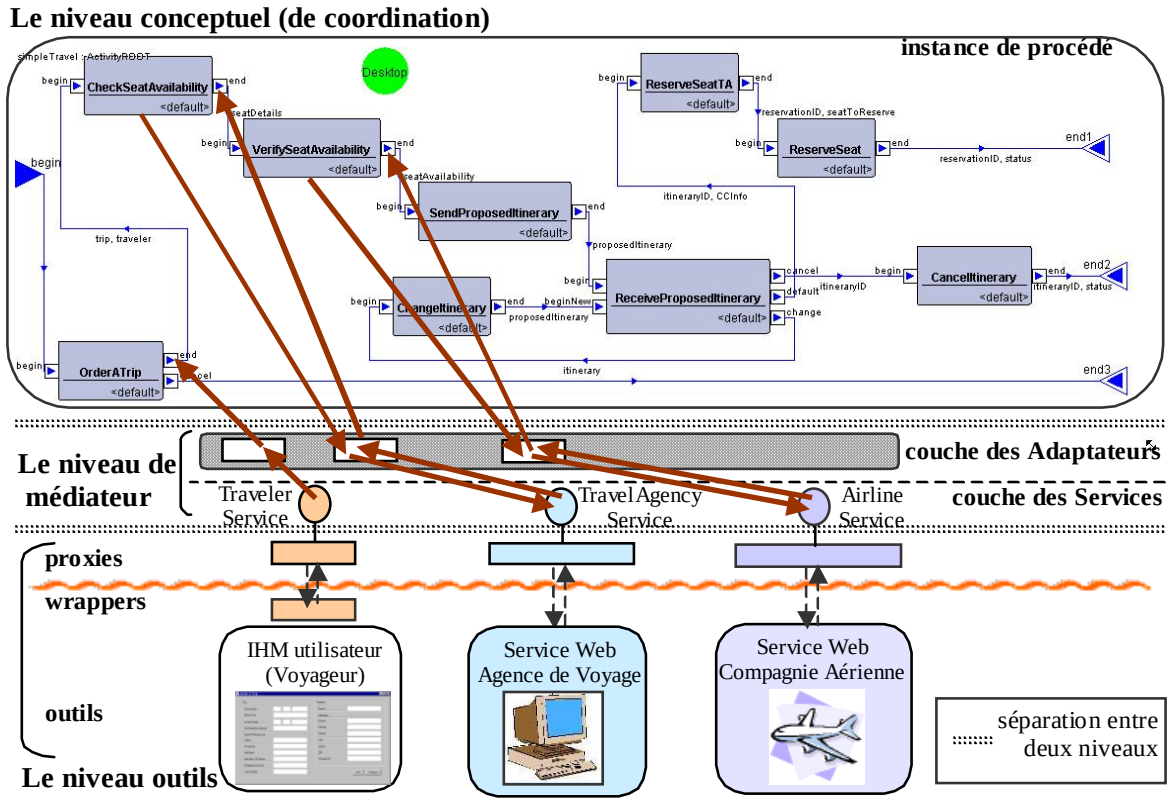


Figure 32 : L'architecture à trois niveaux de l'application Planification de Voyages

Nous avons construit deux services web afin de représenter les deux outils Agence de Voyage et Compagnie Aérienne pour notre application. Nous avons également construit une

application IHM Voyageur représentant le troisième outil de la couche outils de notre application Planification de Voyages. Nous avons ensuite spécifié les services abstraits de ces trois outils pour la couche médiateur. Et nous avons construit les proxies et le wrapper correspondants aux outils et à leurs services. Les Code 4, Code 5, et Code 6, présentent un aperçu des trois services : le service de compagnie aérienne "AirlineService", le service d'agence de voyage "TravelAgencyService" et le service de voyageur "TravelerService".

```
public interface AirlineService {
        public Hashtable getSeatAvailability(Hashtable seatDetails);
        public Vector reserveSeat(Hashtable reservationID, Hashtable seatToReserve);
        public Vector sendReservationNotification(Hashtable reservationID, Hashtable status);
        ...}
```

Code 4 : le service de compagnie aérienne "AirlineService"

```
public interface TravelAgencyService {
        public Hashtable getSeatDetails(Hashtable traveler, Hashtable trip);
        public Hashtable getProposedItinerary(Hashtable seatAvailability);
        public Vector reserveSeat(Hashtable CCInfo, Hashtable itineraryID);
        public Vector sendReservationNotification(Hashtable reservationID, Hashtable status);
        ...}
```

Code 5 : le service d'agence de voyage "TravelAgencyService"

```
public interface TravelerService {
        public void processLaunch();
        public void endOrderATrip(Hashtable travelerAttributes, Hashtable tripAttributes);
        public void cancelOrderATrip();
        public void launchReceiveProposedItinerary (Hashtable proposedItinerary);
        public void launchReceiveReservationNotification (Hashtable reservationID,
                                        Hashtable status);
        public void reserveTickets(Hashtable itineraryID, Hashtable ccInfo);
        ...}
```

Code 6 : le service de voyageur "TravelerService"

4.3. Planification de Voyages : exécution de l'application

Nous avons construit notre application Planification de Voyages de façon à ce que l'instance de procédé se trouvant au niveau conceptuel soit lancée au moment du démarrage de l'outil Voyageur. Ceci ce fait à l'aide de la méthode "processLaunch" du service de voyageur "TravelerService", et à travers un adaptateur transmettant la demande à la couche conceptuelle.

Lorsque le voyageur a fini de remplir les informations concernant son voyage via l'IHM, il appuie sur le bouton OK. A ce moment les informations saisies sont transmises – à travers la

méthode "endOrderATrip" du service "TravelerService" et un adaptateur de la couche médiateur – vers le port de sortie "end" de l'activité "OrderATrip" de la couche conceptuelle. Nous avons choisi que ce port de sortie soit automatique. Cela signifie que dès que les produits attendus dans ce port arrivent, l'activité est terminée et le port de sortie transmet ces produits à l'activité suivante ("CheckSeatAvailability" dans notre exemple). Il n'y a donc pas besoin de terminer l'activité manuellement, comme nous l'avons fait dans l'exemple simplifié de rédaction de document.

Lorsque les produits arrivent à l'activité "CheckSeatAvailability", celle-ci devient active. A ce moment-là, l'adaptateur que nous avons nommé CheckSeatAvailability (Code 7) est démarré. Nous avons en effet spécifié dans cet adaptateur que son démarrage se fait au moment où l'activité CheckSeatAvailability devient active (voir la condition au début de l'adaptateur dans le Code 7).

```
aspect CheckSeatAvailability( int newState ) of Activity {
   when instance.getName( ).equals("CheckSeatAvailability") && instance.isActive( ) ;
   components {
      travelAgency : travelAgencyService;
      processManager : processManagerService; }
   body( JAVA ) {
      Hashtable traveler = processManager.getProductAttributes(instance,"desktop","traveler");
      Hashtable trip = processManager.getProductAttributes( instance, "desktop", "trip" );
      Hashtable seatDetails = travelAgency.getSeatDetails( traveler, trip );
      processManager.addProductInstance( instance, "end", "seatDetails", seatDetails ); } }
```

Code 7 : l'adaptateur "CheckSeatAvailability"

L'adaptateur CheckSeatAvailability récupère les informations stockées sous la forme d'attributs dans les produits traveler et trip du Desktop de l'activité courante CheckSeatAvailability. Il les envoie à l'outil Agence de Voyage et récupère de cet outil le produit seatDetail contenant les détails des places fournies par l'agence de voyage (le nom de la compagnie aérienne choisie par l'agence de voyage, les détails concernant le voyageur, la date du voyage…). Ceci est fait à travers la méthode getSeatDetails du service travelAgency. L'adaptateur envoie alors le produit seatDetails récupéré vers la couche conceptuelle, il l'ajoute dans le port automatique "end" de l'activité courante CheckSeatAvailability (à travers la méthode addProductInstance). Ce port transmet ce produit à l'activité suivante VerifySeatAvailability ce qui termine l'activité courante.

Le procédé de la couche conceptuelle continue à s'exécuter de la même manière à ce que nous venons de décrire. Les changements effectués lors de l'exécution du procédé sont traduits par des appels de services à l'aide des adaptateurs de la couche médiateur. Les outils réagissent sur ces appels et cette réaction est transmise à la couche conceptuelle via les services et les adaptateurs.

Nous remarquons que nous avons utilisé, dans notre exemple d'application de Planification de Voyages, les instructions additionnelles du langage de médiation, ajoutées pour le domaine de procédés (La couche médiateur, Chapitre IV. section 2.2.2). Pour cela, nous avons intégré dans notre application notre Outil d'Extension que nous avons nommé Process Manager, similairement aux outils de l'application : Agence de Voyage, Compagnie Aérienne, et Voyageur. Et ceci en lui attribuant un service : "processManagerService", qui permet de lier cet outil à l'adaptateur qui l'utilise.

De cette façon, nous avons pu utiliser dans notre adaptateur CheckSeatAvailability (Code 7) l'instruction getProductAttributes permettant de récupérer un produit contenu dans un port de l'activité et l'instruction addProductInstance permettant d'ajouter une instance de produit dans un port de l'activité.

4.4. Planification de Voyages : correspondances entre procédé et outils

Dans cet exemple, nous remarquons une correspondance "1 à 1" entre l'activité du procédé de la couche conceptuelle et l'appel d'une méthode d'un outil (envoie-réception). Nous trouvons souvent cette correspondance "1 à 1" dans les langages d'orchestration de services web comme ceux que nous avons présentés dans le chapitre 3.

Lors de la construction, à l'aide de notre plate-forme Mélusine, d'applications de coordination d'outils via un procédé, nous pouvons réaliser une correspondance "1 à n" entre l'activité du procédé et l'appel de méthodes des outils. Ceci car nous considérons que le procédé contient la logique de l'application, et que cette logique n'est pas obligatoirement identique à l'enchaînement des appels de méthodes nécessaires pour implémenter l'application, et c'est en cela qu'une application d'orchestration se distingue des langages de programmation.

Pour cela, il est très important de séparer la couche conceptuelle de la couche de médiation, et de la couche d'outils, non seulement pour des raisons techniques ou pour des raisons de flexibilité, mais surtout parce que ces trois couches se distinguent fortement du point de vue conceptuel :

- La couche conceptuelle contient un modèle exécutable, décrivant la logique de l'application. Cette logique est exprimée par des concepts propres au domaine de l'application ;
- La couche de médiation traduit chaque changement et chaque événement important se passant dans la couche conceptuelle, par une suite d'appels de méthodes des outils ;
- La couche contient des outils offrant un certain nombre de méthodes. Ces outils sont indépendant et ne connaissent pas les deux couches : conceptuelle et médiation.

Ainsi, notre application Planification de Voyages, peut être construite d'une façon différente, en utilisant la correspondance "1 à n" entre l'activité du procédé et l'appel des méthodes d'outils.

Ainsi, une fois que l'outil Voyageur a transmis les informations concernant le voyage et le voyageur à l'activité de démarrage pour commander un voyage : "OrderATrip", ces informations sont envoyées à l'activité "CheckSeatAvailability". Cette activité signifie qu'il faut vérifier les disponibilités de vols vers la destination demandée. Elle sera alors traduite (à l'aide de l'adaptateur : Code 8 et Figure 33) par :

(a) un appel vers l'agence de voyage, à qui sont transmises les informations concernant le voyage et le voyageur, et qui choisit une compagnie aérienne ayant des vols vers cette destination ;

(b) puis un appel vers cette compagnie aérienne, pour lui demander les disponibilités de vols vers cette destination ;

(c) et finalement par un nouvel appel vers l'agence de voyage pour lui envoyer la liste des vols disponibles retournée par la compagnie aérienne. Celle-ci choisit un itinéraire, afin de le proposer au voyageur. Cet itinéraire choisi sera alors ajouté dans le port automatique "end" de l'activité courante "CheckSeatAvailability".

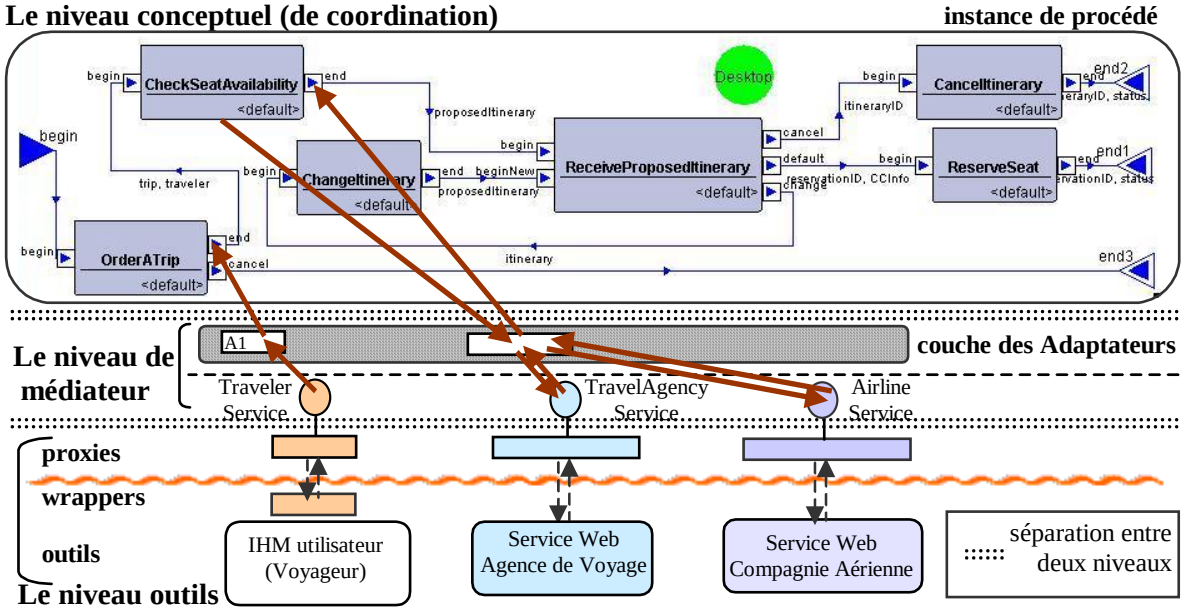


Figure 33 : L'application Planification de Voyages en utilisant la correspondance "1 à n"

En utilisant la correspondance "1 à n", nous avons pu fusionner trois activités de l'ancienne version de l'application : "`CheckSeatAvailability`", "`VerifySeatAvailability`", et "`SendProposedItinerary`" (Figure 32) en une seule, contenant la logique de ces trois activités rassemblées.

De la même menière, nous pouvons fusionner les deux activités ReserveSeatTA et ReserveSeat de l'ancienne version de l'application (Figure 32) en une seule activité ReserveSeat. Car logiquement, cette activité signifie la réservation du vol. Cette demande de réservation doit être transmise à l'agence de voyage et à la compagnie aérienne afin d'effectuer la réservation. L'adaptateur de cette activité peut se charger de transmettre

cette demande aux deux outils. Nous n'avons donc pas le besoin d'exprimer cette demande de réservation en deux activités car du point de vue conceptuel elle ne représente qu'un seul concept.

```
aspect CheckSeatAvailability( int newState ) of Activity {
  when instance.getName( ).equals("CheckSeatAvailability") && instance.isActive(  ) ;
  components {
    travelAgency : travelAgencyService;
    airline : airlineService;
    processManager : processManagerService; }
  body( JAVA ) {
    Hashtable traveler = processManager.getProductAttributes(instance,"desktop","traveler");
    Hashtable trip = processManager.getProductAttributes( instance, "desktop", "trip" );
    Hashtable seatDetails = travelAgency.getSeatDetails(traveler, trip);
    Hashtable seatAvailability= airline.getSeatAvailability(seatDetails);
    Hashtable proposedItinerary = travelAgency.getProposedItinerary( seatAvailability );
    processManager.addProductInstance(instance,"end","proposedItinerary",proposedItinerary);}}
```

Code 8 : l'adaptateur "CheckSeatAvailability" en utilisant la correspondance "1 à n"

4.5. Planification de Voyages : aller un peu plus loin…

Nous avons présenté dans cette section notre application Planification de Voyages qui vise à réserver des billets d'avion, pour effectuer un voyage.

Nous pouvons aller un peu plus loin en construisant une application qui gère non seulement la réservation des vols mais en plus d'autres services comme la réservation de logement, la location de voiture ou de vélo…

Pour cela, nous pouvons construire notre procédé correspondant à la logique de l'application de manière hiérarchique. Nous construisons un procédé global exprimant l'enchaînement des services principaux que gère notre application. Chaque activité de ce procédé global peut contenir un "sous-procédé" décrivant la logique du service que représente cette activité du procédé global. La Figure 34 illustre ce procédé global et le sous-procédé contenu dans l'activité "Réservation vol" qui décrit la logique du service de réservation de billets d'avions que nous venons de présenter.

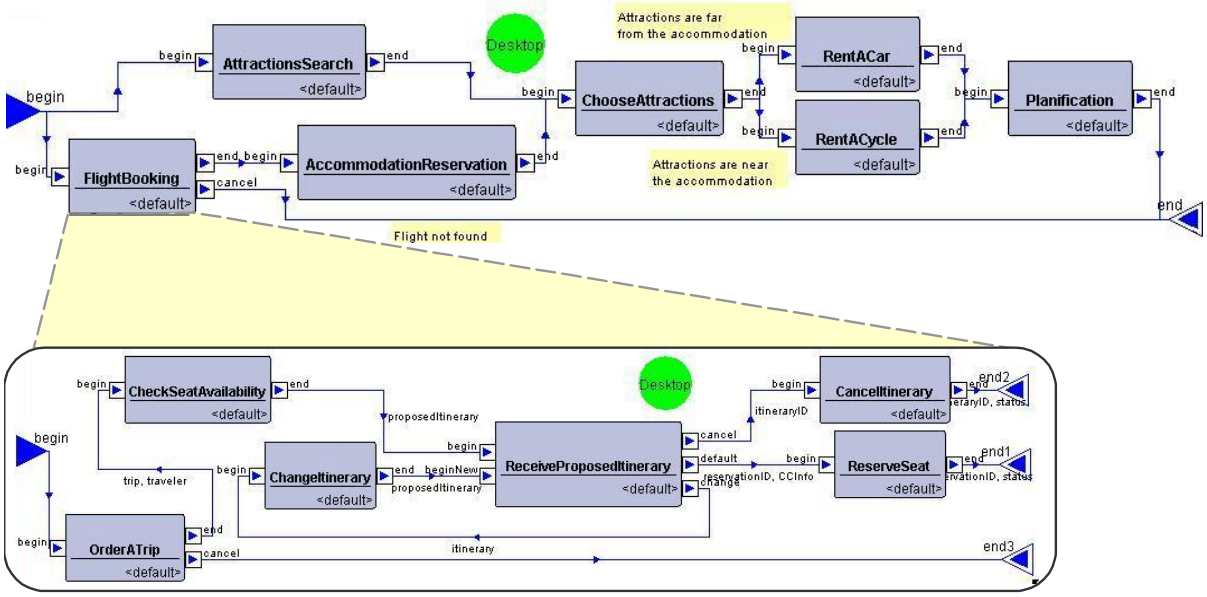


Figure 34 : Le procédé global Planification de Voyages et le sous-procédé "Réservation vol"

Ce procédé global, avec ses sous procédés, formera la couche conceptuelle de notre application. Les changements concernant les concepts de cette couche conceptuelle, pendant l'exécution, seront interprétés par les adaptateurs, afin de les traduire en appels de méthodes des outils participants à l'application, comme nous l'avons expliqué précédemment.

4.6. Planification de Voyages : conclusion

Nous avons présenté dans cette section notre application Planification de Voyages, qui est une application d'orchestration de deux services web et d'un outil offrant une IHM voyageur s'exécutant à distance. Pour définir cette application, nous nous sommes inspirés d'un exemple contenu dans la spécification du langage WSCI [AAF 02].

Nous avons construit cette application à l'aide de notre plate-forme Mélusine. L'application Planification de Voyages a donc une architecture à trois couches (conceptuelle, médiateur, et outils).
La couche outils contient les trois outils participant à cette application : le service web Agence de Voyage, le service web Compagnie Aérienne et l'application IHM utilisateur. La couche conceptuelle contient le procédé de coordination de ces trois outils. La couche médiateur fait le lien entre ces deux couches à travers ses adaptateurs et ses services.

La définition d'un méta-modèle de procédé ne contenant qu'un minimum de concepts permet de construire des procédés de coordination compréhensible et facile à manipuler pour le concepteur du logiciel.

L'utilisation de ces trois couches permet de séparer la logique de coordination de l'application, de son implémentation. Isoler la logique de coordination dans une couche conceptuelle permet d'avoir une vision globale claire du procédé de coordination. Cela offre

également beaucoup de flexibilité pour faire évoluer ce procédé de coordination. Nous pouvons par exemple ajouter une activité et la relier à deux autres activités de notre procédé, et cela très facilement. Nous pouvons ensuite lier cette activité à un outil à travers un adaptateur et un service afin d'exécuter la tâche qu'exprime cette activité. Nous pouvons donc modifier la logique et l'exécution de notre logiciel sans avoir à faire de grandes modifications au niveau du code source.

La couche médiateur, qui lie les deux couches conceptuelle et outils, donne beaucoup de flexibilité à l'application. Ceci car en identifiant une couche de services, contenant la description des services abstraits de l'application, nous pouvons substituer un outil par un autre. Il suffit que cet outil corresponde à ce service abstrait. Cette correspondance entre l'outil et le service est faite à l'aide des proxies et des wrappers.

La couche médiateur contient également les adaptateurs. Nous avons construit un langage extensible de médiation. Et nous avons vu dans cette application d'orchestration à l'aide d'un procédé, l'utilisation de notre langage d'adaptateurs, en utilisant une extension pour le domaine de procédés.

Nous avons vu également que nous pouvons, à travers cette architecture, réaliser des correspondances "1 à n" entre l'activité du procédé et l'appel de méthodes d'outils. Ce qui différencie notre orchestration d'outils des langages de programmation.

Nous voyons à travers cet exemple d'orchestration de services que notre architecture, présentée dans ce chapitre, répond à la plupart des besoins que nous avons identifié dans les langages de coordination de services web (Chapitre III).

Cependant, il reste un point auquel nous n'avons pas encore répondu. Ce point concerne l'évolution des concepts de notre logiciel de coordination. Car, comme nous l'avons dit précédemment, il y aura toujours de nouveaux concepts à intégrer pour réaliser de nouveaux scénarios et pour répondre à de nouveaux besoins particuliers. Néanmoins, cette évolution de concepts du logiciel de coordination doit être faite tout en gardant la simplicité du méta-modèle et la facilité de manipulation des concepts et de construction de logiciel utilisant ces nouveaux concepts. Nous allons présenter dans le chapitre suivant notre approche qui propose une solution efficace à ce problème d'évolution de concepts du logiciel.

5. Conclusion

Nous avons présenté dans ce chapitre la plate-forme Mélusine qui e été développée dans notre équipe. Nous définissons à travers cette plate-forme, une méthodologie de construction de logiciels. Ceci en définissant une architecture à trois couches. Cette architecture permet d'identifier un niveau conceptuel du logiciel, contenant le noyau de concepts métier du logiciel. Elle permet de séparer cette couche conceptuelle de la couche contenant l'implémentation du logiciel. Ces deux couches sont liées à travers une troisième couche appelée médiateur et qui contient les adaptateurs et les services.

L'architecture à trois couches offre beaucoup d'avantages comme nu avons pu l'illustrer dans ce chapitre. Parmi ces avantages, nous avons la définition d'un méta-modèle métier minimal ne contenant que les concepts métier de base. Nous avons aussi la possibilité de substituer facilement un outil par un autre, du moment où il implémente le même service, et ceci sans modifier le niveau conceptuel ni le niveau médiateur.

Mélusine est à la fois un environnement de conception permettant de construire des applications respectant cette architecture et un environnement d'exécution offrant un moteur pour exécuter ces applications. L'utilisation de notre plate-forme pour exécuter les applications offre également beaucoup d'avantages. Elle offre en effet la persistance et la reprise sur pannes, elle contient des outils de visualisation permettant la supervision de l'exécution et elle se charge de gérer l'interopérabilité entre les différentes couches conceptuelle, médiation et outils de l'application.

Notre plate-forme Mélusine peut être également vue comme un environnement de procédé. Elle permet de concevoir et d'exécuter des systèmes dirigés par un procédé, se trouvant dans la couche conceptuelle. Nous pouvons ainsi construire toute sorte de système dirigé par un procédé comme ceux présentés dans l'état de l'art de ce document de thèse : des systèmes de coordination d'outils via un procédé, des systèmes de planification et gestion du cycle de vie d'un projet, des systèmes de développement logiciel, des systèmes de contrôle et d'assistance à l'évolution de logiciels, des systèmes de coordination entre membres d'un groupe...

Nous avons présenté dans la section 7 du chapitre 2, un tableau résumant les défauts concernant les procédés de coordination et les systèmes de gestion de coordination existants ainsi que les besoins qu'il faut prendre en compte dans les systèmes de gestion de coordination par procédés. Nous rappelons ces besoins dans la Table 8.

Besoins des Procédés de Coordination
Procédés exécutables de coordination, centré activités.
▪ Simplifier les concepts ; ▪ Se baser sur un minimum de concepts ; ▪ Permettre l'évolution en permettant l'introduction de nouveau concepts.
Formalisme de haut niveau (décrit la logique de l'application).
Procédé flexible (facile à changer et à faire évoluer).
Couplage faible (facilité de substitution d'un service par un autre).
Coordonner des outils divers (services web, programmes locaux ou distants, composants, COTS, etc.).

Besoins des Systèmes de gestion de Coordination
Environnement : ▪ Performant ; ▪ Gérant la reprise sur pannes ; ▪ Contenant des outils d'aide pour l'utilisateur (outils d'analyse, de test et de débogage) ; ▪ Gérant l'interopérabilité entre le système et les outils.
Un logiciel spécialisé, s'occupant de l'ensemble de nos besoins.

Table 8 : Les besoins des procédés de coordination, et des systèmes de gestion de coordination

Notre plate-forme Mélusine permet de construire et d'exécuter des applications de coordination basées sur procédés centrés activités. Ce procédé est basé sur le méta-modèle d'APEL qui est un méta-modèle de procédé simple, minimal et ne contenant que les concepts de base.

Mélusine contient l'éditeur graphique APEL qui facilite la construction des modèles de procédés conformes au méta-modèle d'APEL. Ceci constitue un formalisme de haut niveau permettant de décrire la logique de l'application.

L'éditeur graphique APEL permet également de modifier facilement les procédés construits à l'aide de cet éditeur, ce qui rend ces procédés très flexibles.

L'architecture à trois niveaux des applications construites à l'aide de notre plate-forme Mélusine permet de séparer la logique de l'application contenue dans la couche conceptuelle de son implémentation en terme d'outils participant à l'application.

Cette architecture offre un couplage faible entre le procédé et les outils qu'il coordonne. Elle permet de substituer facilement un outil par un autre, du moment où il implémente le même service, et ceci sans modifier le niveau conceptuel ni le niveau médiateur.

Le niveau outil des applications construites à l'aide de notre plate-forme Mélusine permet d'utiliser toute sorte d'outils basés sur des technologies diverses. La coordination d'outils construite à l'aide de Mélusine n'est donc pas réservée à des technologies d'outils particulières.

Mélusine est également un environnement d'exécution performant pour les applications de coordination d'outils via un procédé. Cet environnement couvre la persistance et la reprise sur panne et contient des outils d'aide pour l'utilisateur (des outils de visualisation, de contrôle de cohérence...). L'environnement se charge également de gérer l'interopérabilité entre les différentes couches conceptuelle, médiation, et outils de l'application.

Ainsi, Mélusine nous permet de construire des logiciels spécialisés de coordination d'outil, à l'aide de procédés, s'occupant de l'ensemble de nos besoins.

Un seul point n'a pas encore été traité dans ce chapitre, il concerne l'évolution des concepts de notre logiciel de coordination. Car il y aura certainement des évolutions concernant les concepts des logiciels que nous avons construits via Mélusine. Ces évolutions peuvent être dues au temps ou au besoin de répondre à de nouvelles exigences. Néanmoins, l'évolution de ces concepts doit se faire en gardant la simplicité du méta-modèle et la facilité de manipulation des concepts et de construction de logiciel utilisant ces nouveaux concepts.

Nous allons présenter dans le chapitre suivant notre approche qui propose une solution efficace à ce problème d'évolution de concepts du logiciel.

Chapitre V.
Mélusine : une plate-forme extensible et évolutive

1. Introduction

Nous avons présenté dans le chapitre précédant notre plate-forme Mélusine, qui est un environnement de conception et d'exécution de systèmes logiciels. Nous avons défini à travers cette plate-forme une méthodologie de construction de logiciels. Ceci en définissant une architecture à trois niveaux : le niveau conceptuel, le niveau médiateur et le niveau outils. Nous avons présenté les avantages de cette architecture à trois niveaux.

Nous avons vu que la conception de logiciels à l'aide de la plate-forme Mélusine se fait en deux étapes : la construction de la Machine Virtuelle du Domaine d'Applications en premier et ensuite la construction de l'Application du Domaine à l'aide de cette Machine Virtuelle.

La Machine Virtuelle implémente le méta-modèle du domaine d'applications. Suivant la philosophie de Mélusine ce méta-modèle doit être minimal ne contenant que les concepts métier formant les éléments de base.

Lors de la construction d'une application via Mélusine, à l'aide de la Machine Virtuelle du domaine de cette application, il nous arrive d'avoir besoin d'un modèle plus complexe contenant des concepts additionnels ou des concepts plus détaillés que ce que l'on trouve dans le méta-modèle minimal de la Machine Virtuelle. Il arrive aussi que des applications aient à évoluer ayant pour conséquence l'ajout de nouveaux concepts inexistants dans le méta-modèle minimal.

Néanmoins, l'ajout de ces concepts est souvent spécifique à une ou à un nombre limité d'applications d'un domaine. La plupart du temps chaque application pour un domaine a besoin de ses propres concepts additionnels. L'ajout de tous ces concepts additionnels au méta-modèle du domaine serait une mauvaise solution. En effet cela encombrerait le méta-modèle de concepts ne servants qu'une ou deux fois (comme cela a pu être le cas dans la version V4 d'APEL).

Une meilleure approche, que nous proposons dans ce travail de thèse, est de se baser sur le méta-modèle simple et minimal du domaine de l'application et d'offrir des mécanismes d'extension afin d'ajouter de nouvelles fonctionnalités à l'application.

Dans ce chapitre, nous expliquons les différentes façons d'étendre et de faire évoluer les applications construites à l'aide de notre plate-forme Mélusine, en étendant le modèle de l'application se trouvant dans le niveau conceptuel.

2. Evolution des applications construites à l'aide de Mélusine

Nous distinguons quatre formes d'évolutivité que nous proposons dans notre approche :
1. L'extension sémantique (features) ;
2. L'extension de méta-modèle ;
3. Le raffinement ; et
4. La composition.

Ces quatre formes d'évolutivité peuvent être combinées selon les besoins.

2.1. L'extension sémantique (features)

La première forme d'évolutivité que nous proposons est l'extension sémantique.

2.1.1. Définition de l'extension sémantique

L'*extension sémantique* permet de modifier la sémantique (le comportement) d'un concept du méta-modèle courant. Pour cela, nous utilisons ce que nous appelons dans notre approche "les features" (les options). Ainsi, une option contient des interprétations sémantiques de plusieurs concepts du méta-modèle. L'utilisateur de l'application construite à l'aide de notre plate-forme Mélusine peut alors choisir les options couvrant les fonctionnalités qu'il souhaite avoir dans son environnement. Il peut également définir de nouvelles options contenant son interprétation des concepts du méta-modèle dont il souhaite modifier la sémantique.

2.1.2. Exemple de l'extension sémantique : notre logiciel de rédaction de document

Dans notre logiciel de rédaction de document présenté dans le chapitre 4, nous pouvons par exemple souhaiter étendre le concept d'activité du méta modèle de procédé afin d'ajouter la notion de traces d'exécution à ce concept. Cette notion de traces pourrait se traduire dans le logiciel par l'annotation dans un fichier de certaines informations concernant la date, l'heure, le nom de l'activité qui vient de démarrer ou de se terminer ainsi que le nom du responsable de cette activité.

Pour réaliser cette extension sémantique, nous ajoutons un adaptateur Traces (Code 9), un service TraceService (Code 10) dans la couche du médiateur et un outil Traces (Code 11) dans la couche des outils. Pour lier le service et l'outil à notre logiciel, nous utilisons l'éditeur graphique de notre plate-forme Mélusine : FedeEditor (Figure 35). Cet éditeur nous permet

également de choisir, en sélectionnant dans une liste, le concept de la couche conceptuelle du logiciel que nous souhaitons étendre sémantiquement et d'écrire l'adaptateur liant ce concept à l'outil à travers son service (Figure 36). Cet éditeur nous offre également des facilités de vérification et de compilation des adaptateurs construits.

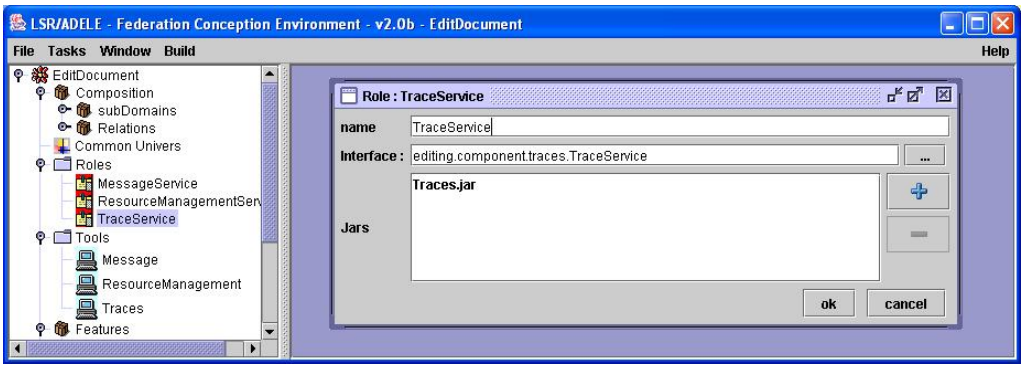


Figure 35 : Lier le service TraceService au logiciel EditDoc à l'aide de FedeEditor

La Figure 35 et la Figure 36 illustrent l'éditeur graphique FedeEditor de notre plate-forme Mélusine en affichant les détails de notre logiciel de rédaction de document EditDocument. Nous remarquons dans ces deux figures que notre logiciel de rédaction de document contient les trois services que nous avons présentés : MessageService, RessourceManagementService et TraceService ainsi que les trois outils qui les implémentent : Message, RessourceManagement et Trace.

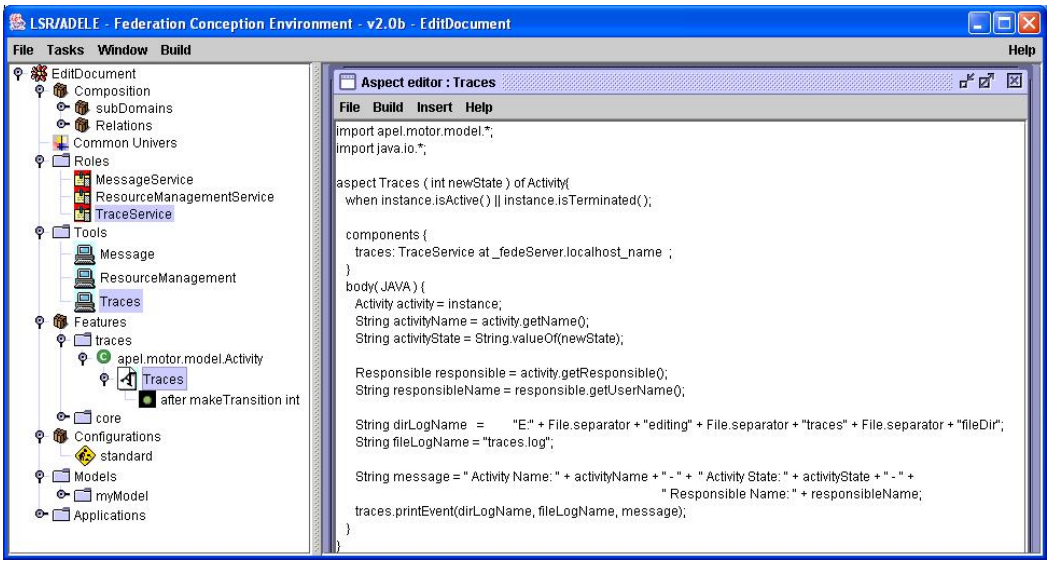


Figure 36 : L'ajout d'un adaptateur, d'un service, et d'un outil à l'aide de l'éditeur "FedeEditor"

Nous précisons dans notre adaptateur (Code 9) qu'il doit s'exécuter à chaque fois qu'une activité du procédé démarre ou se termine. Il rassemble alors les informations concernant le nom de l'activité, son état et le nom de son responsable. Il envoie ces informations à l'outil Traces sous forme d'un message à travers la méthode printEvent du service TraceService que cet outil implémente. Pour cela il précise le chemin et le nom du fichier dans lequel seront

inscrites ces informations. L'outil Traces exécute alors cette méthode printEvent qui inscrit en premier la date et l'heure précise de son exécution dans le fichier spécifié et y inscrit ensuite le message envoyé par l'adaptateur. La Figure 37 illustre cette extension sémantique du concept d'activité.

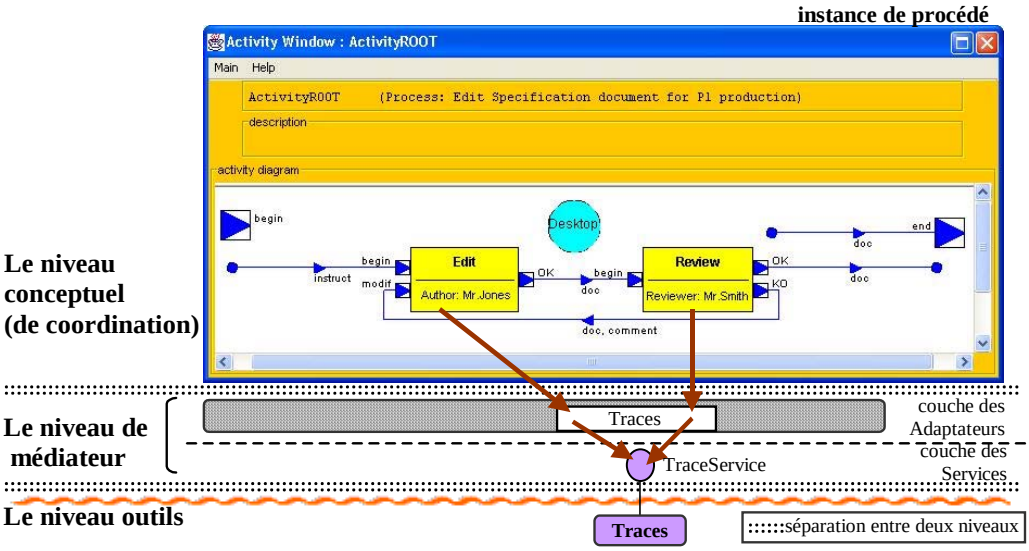


Figure 37 : Extension sémantique du concept d'activité afin d'ajouter la notion de traces

```
import apel.motor.model.*;
import java.io.*;
aspect Traces ( int newState ) of Activity{
   when instance.isActive( ) || instance.isTerminated( );
   components {  traces : TraceService ;  }
   body( JAVA ) {
      Activity activity = instance;
      String activityName = activity.getName();
      String activityState = String.valueOf(newState);
      Responsible responsible = activity.getResponsible();
      String responsibleName = responsible.getUserName();
      String dirLogName = "E :" + File.separator + "editing" + File.separator +
                                    "traces" + File.separator + "fileDir";
      String fileLogName = "traces.log";
      String message = " Activity Name : " + activityName + " - " + " Activity State : " +
                       activityState + " - " + " Responsible Name : " + responsibleName;
      traces.printEvent(dirLogName, fileLogName, message);   } }
```

Code 9 : L'adaptateur Traces[52]

[52] Le paramètre newState est un paramètre de la méthode makeTransition s'exécutant lorsque l'activité change d'état. Il correspond au nouvel état de cette activité (init, ready, active, terminated, aborted, suspended).

```
public interface TraceService {
     public void printEvent( String dirLogName, String fileLogName, String message ); }
```

Code 10 : Le service TraceService

```
import java.io.*;
import java.util.Date;
public class Traces implements TraceService {
    private static FileWriter out;
    // Ouvre le fichier
    private static void init(String dirLogName, String fileLogName) {
        File dirLog = new File( dirLogName );
        if( !dirLog.exists() ) dirLog.mkdirs();
        try { out = new FileWriter( dirLogName + File.separator + fileLogName, true ); }
        catch( Exception e ) { e.printStackTrace(); } }
    // Enregistre un événement dans le fichier
    public void printEvent( String dirLogName, String fileLogName, String message ) {
        init(dirLogName, fileLogName);
        try { out.write( "[" + new Date( ).toString( ) + "] : " + message + "\r\n" );
              out.flush(); }
        catch( Exception e ) { e.printStackTrace(); } }  }
```

Code 11 : L'outil Traces

2.1.3. L'extension sémantique : une extension optionnelle

Lorsque nous construisons des extensions sémantiques d'un logiciel construit à l'aide de notre plate-forme Mélusine, ces extensions forment des options. L'utilisateur du logiciel pourra choisir d'exécuter ce logiciel avec cette option ou sans elle.

Dans notre logiciel de rédaction de document, l'extension sémantique que nous avons construit correspond à l'option traces. Cette option contient un seul adaptateur (l'adaptateur Traces que nous avons décrit précédemment). L'éditeur graphique de notre plate-forme Mélusine FedeEditor (Figure 38) nous permet de créer l'option et de spécifier l'adaptateur (ou les adaptateurs) qui réalise cette extension sémantique. Les options définies dans le logiciel sont présentés dans un tableau. L'utilisateur du logiciel pourra choisir les options qu'il souhaite dans cette exécution du logiciel.

Dans la Figure 38, qui illustre les options du logiciel EditDocument, nous remarquons que ce logiciel contient deux options :

1) core constituant le cœur de l'application. Il contient les cinq adaptateurs que nous avons présentés dans le chapitre 4 et qui réalisent le logiciel de rédaction de document ;
2) trace qui forme l'extension sémantique que nous avons présenté dans cette section. Cette option contient l'adaptateur Traces interceptant la méthode makeTransition de la classe Activity contenue dans le paquetage apel.motor.model.

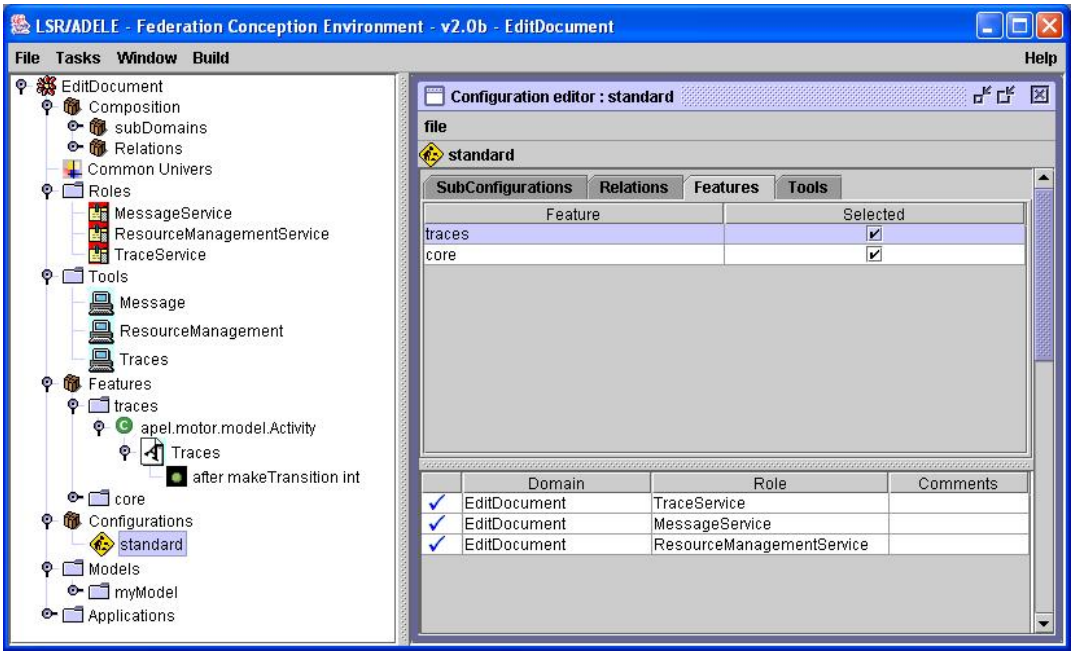


Figure 38 : Les options du logiciel EditDocument affichés dans l'éditeur FedeEditor

Dans la configuration portant le nom standard, les deux options sont sélectionnées. Ceci indique que l'exécution de cette configuration inclus les deux options : core et trace. Nous pouvons, à l'aide de FedeEditor, définir plusieurs configurations pour le même logiciel et choisir celle qui sera exécutée.

Il est important de noter que l'option ne modifie pas le méta-modèle courrant et plus particulièrement qu'elle n'ajoute pas de nouveaux concepts au méta-modèle.

2.1.4. Evaluation

Nous constatons avec cet exemple la facilité de réaliser une extension sémantique. Pour le concepteur du logiciel, l'extension sémantique d'un concept se réduit à :

- construire l'outil contenant les fonctions souhaitées (l'outil Traces dans notre exemple) ;
- décrire l'interface (le service) de cet outil (TraceService dans notre exemple) ;
- lier le service et l'outil au logiciel en sélectionnant l'interface et la classe de l'outil qui l'implémente dans l'interface graphique correspondante de l'éditeur FedeEditor (Figure 35) ;
- sélectionner, dans une liste de FedeEditor, le concept du logiciel que l'on souhaite étendre sémantiquement ;
- écrire dans FedeEditor l'adaptateur liant le concept du logiciel au nouvel outil à travers son service (l'adaptateur Traces dans notre exemple, Figure 36) ;
- et finalement spécifier dans FedeEditor une nouvelle option contenant ce nouvel adaptateur (Figure 38).

FedeEditor se charge alors de compiler le tout, afin de le rendre exécutable.

Nous mentionnons ici que le code présenté (Code 9, Code 10, et Code 11) forme la totalité de ce que doit écrire le concepteur du logiciel pour réaliser notre extension sémantique trace. Nous remarquons le peu de ligne de code à écrire pour réaliser le lien entre le concept entendu sémantiquement et l'outil (le code de l'adaptateur et du service). Cependant, la taille de l'outil varie suivant les fonctions que nous souhaitons qu'il réalise. Il est également possible d'utiliser un outil existant à condition qu'il offre une interface contenant les méthodes que nous souhaitons utiliser. Si cet outil ne contient pas d'interface, il faudra alors lui construire un wrapper (une enveloppe) afin d'offrir cette interface. La construction de ce wrapper peut être plus ou moins complexe suivant l'outil et les fonctions que nous attendons de lui (si ces fonctions font partie des méthodes de l'outil ou qu'il faut combiner plusieurs méthodes pour les réaliser).

Nous constatons aussi la facilité d'utilisation de l'extension sémantique. En effet, pour l'utilisateur du logiciel, il lui suffit de cocher (ou décocher) la case correspondante à l'option qu'il souhaite ajouter (retirer) de l'exécution du logiciel.

2.2. Extension de méta-modèle

La deuxième forme d'évolutivité que nous présentons dans ce chapitre est l'extension de méta-modèle.

2.2.1. Définition de l'extension de méta-modèle

Cette extension consiste à ajouter de nouveaux concepts au méta-modèle. Ces nouveaux concepts peuvent être fonctionnels et/ou non fonctionnels. Ainsi, suivant les fonctionnalités que l'on souhaite avoir dans l'environnement que l'on construit, nous pouvons à l'aide de l'extension de méta-modèle ajouter tous les concepts dont on a besoin.

Nous nous sommes concentrés, dans notre travail, sur l'extension de méta-modèle par des concepts fonctionnels. L'extension de méta-modèle par des concepts non-fonctionnels, tels que les transactions, a été identifiée mais n'a pas encore pu être approfondie. Elle fait partie des perspectives de notre travail que nous présentons dans la section 4 du Chapitre VI.

2.2.2. Construction de l'extension de méta-modèle

La construction d'une extension de méta-modèle se passe de manière similaire à la construction du logiciel à l'aide de la plate-forme Mélusine. Cette construction se fait en deux étapes. La première étape consiste à construire une Machine Virtuelle ne contenant que les nouveaux concepts. Dans la deuxième étape, nous combinons cette Machine Virtuelle avec la Machine Virtuelle du Domaine d'Applications, afin de construire notre Extension de méta-modèle.

Nous appelons cette nouvelle Machine Virtuelle : la Machine Virtuelle de l'Extension. Elle implémente le méta-modèle, décrivant les concepts que nous souhaitons ajouter au méta-

modèle de notre domaine d'applications et les relations entre ces concepts dans le cadre de cette extension. Ainsi, à l'aide de cette Machine Virtuelle d'Extension, nous pouvons construire notre modèle d'extension conformément à ce méta-modèle. La Machine Virtuelle d'Extension contient également un moteur permettant d'exécuter une instance de ce modèle et de réifier ses concepts.

2.2.3. Exemple de l'extension de méta-modèle : notre logiciel de rédaction de document

Dans notre exemple de rédaction de document, nous pouvons constater pendant l'utilisation de notre application le besoin d'un concept d'exception. Nous pouvons alors construire une extension de méta-modèle consistant à ajouter le concept d'exception à notre méta-modèle de procédé. Pour cela, nous définissons tout d'abord le méta-modèle d'exception.

1) Définition du méta-modèle de notre extension "Exception"

Nous spécifions dans ce méta-modèle le concept d'exception comme étant spécialisé en trois sous-classes : l'alarme de deadline (date limite), l'exception de condition et l'exception de cohérence. Nous spécifions également dans ce méta-modèle le concept de deadline (date limite), et le concept de condition (Figure 39).

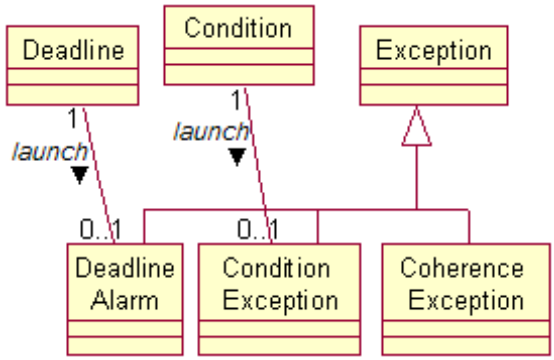


Figure 39 : le méta-modèle d'exception

Le but de ces concepts est de permettre de définir, dans un logiciel basé sur un modèle de procédé, des deadlines pour les activités et des conditions à vérifier dans les ports d'entrées et de sorties des activités. La deadline d'une activité permettra de déclencher une exception du type alarme de deadline si l'activité n'est pas terminée avant cette date de fin. La condition ajoutée dans un port d'entrée ou de sortie d'une activité permettra de faire des vérifications sur les produits entrants ou sortants de l'activité. Elle permettra de déclencher une exception du type exception de condition dans le logiciel si la condition n'est pas vérifiée.

2) Construction de la Machine Virtuelle du domaine de l'extension

Après avoir défini le méta-modèle de notre extension Exception, nous construisons la Machine Virtuelle du domaine de cette extension. Il nous faut alors construire un moteur permettant d'exécuter les instances de modèle d'extension construits conformément au méta-modèle d'extension défini précédemment et de réifier leurs concepts.

Dans notre exemple, nous choisissons de construire un moteur simple contenant une classe ExceptionManager. Cette classe permet de gérer les instances de concepts du modèle d'extension construit : créer une deadline, lui attribuer une valeur (une date), la supprimer... (Code 12). A l'aide de cette classe, nous pourrons réaliser une exécution virtuelle de notre modèle d'extension, en réifiant les concepts de ce modèle. Cette exécution virtuelle constituera le niveau conceptuel de l'extension de notre application au cours de son exécution.

```
public class ExceptionManager {
  public void createDeadline(String activityName) {...}
  public void createDeadline(String activityName, int year, int month, int day,
                      int hours, int minutes, int seconds){...}
  public void saveDeadlineDate(String activityName, int year, int month, int day,
                      int hours, int minutes, int seconds){...}
  public void deleteDeadline(String activityName) {...}
  ...
}
```

Code 12 : La classe ExceptionManager

3) Les relations entre le domaine principal et l'extension

Nous définissons ensuite les relations entre le domaine principal (domaine de procédés dans notre exemple) et le domaine de l'extension (domaine d'exception). Nous présentons ici les relations entre ces deux domaines pour les trois niveaux : méta-modèle, modèle et instance.

Niveau méta-modèle

Nous ajoutons donc les nouveaux concepts dans le niveau méta-modèle. Nous lions le concept d'exception de cette extension au concept d'activité du méta-modèle de procédé par une dépendance orientée de "exception" vers "activity" (Figure 40). Cela exprime, au niveau du méta-modèle, que l'exception du modèle d'extension dépend de l'activité du modèle de procédé au niveau modèle du logiciel. Ainsi, au niveau modèle, une exception déclenchée sera nécessairement liée à une activité du modèle de procédé.

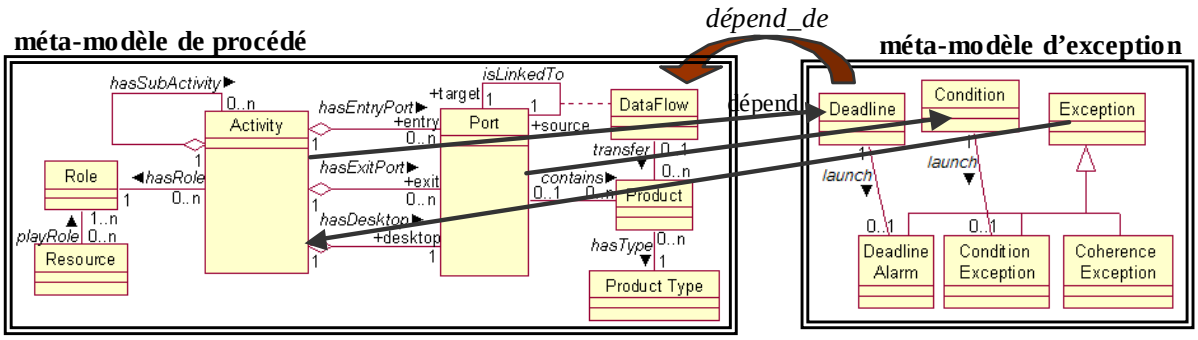


Figure 40 : le niveau méta-modèle

Nous lions également les concepts "port", resp. "activity" du méta-modèle de procédé par un lien de dépendance aux concepts "condition", resp. "deadline" du méta-modèle d'extension. Cela pour exprime au niveau modèle qu'un port du modèle de procédé pourra dépendre d'une condition du modèle d'exception qu'il faudra vérifier et qu'une activité du modèle de procédé pourra dépendre d'une date limite pour cette activité.

Niveau modèle

Le niveau modèle du logiciel contient à la fois le modèle de procédé de ce logiciel et le modèle d'extension qui correspond dans notre exemple au modèle d'exception. Dans la Figure 41 nous avons choisi d'attribuer une date de fin à l'activité "Review" de notre modèle de procédé. Le modèle d'extension contient alors une date limite et une alarme pour cette activité.

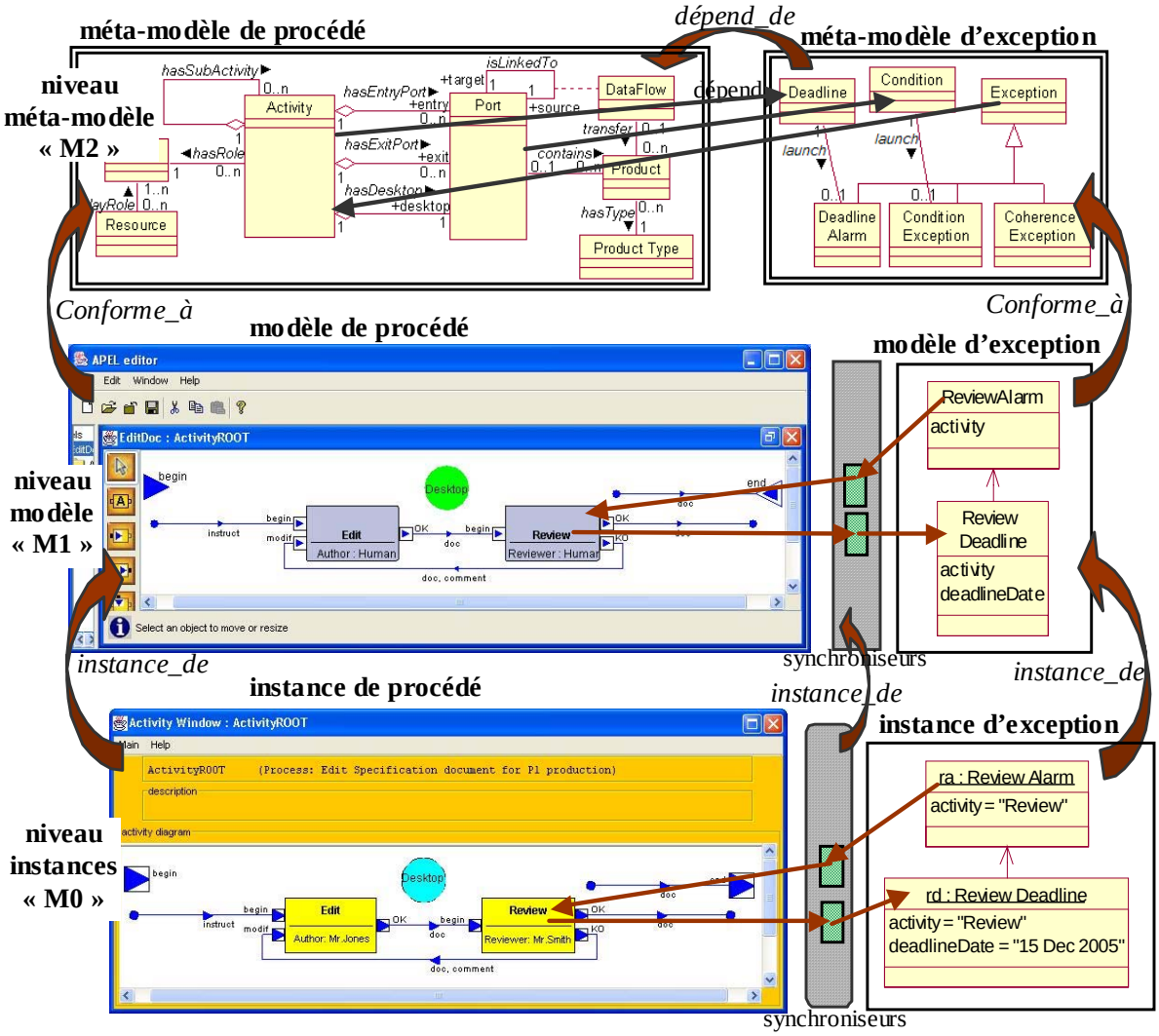


Figure 41 : Extension de méta-modèle

Les dépendances que nous avons exprimées au niveau méta-modèle entre le concept d'exception et le concept d'activité et entre le concept d'activité et le concept de deadline

seront traduits au niveau modèle par des synchroniseurs d'extension. Un synchroniseur est similaire à un adaptateur sauf qu'il a pour fonction de faire communiquer deux concepts du même niveau conceptuel alors que l'adaptateur permet de relier un concept du niveau conceptuel avec des services de la couche médiateur.

Dans la Figure 41, nous déclarons au niveau modèle deux synchroniseurs d'extension. Nous définissons un synchroniseur pour décrire la dépendance entre l'exception "ReviewAlarm" de type "DeadlineAlarme" et l'activité "Review" du modèle de procédé. Ce synchroniseur traduit la relation "*exception – activité*" du méta-modèle. La deuxième dépendance, "*activité – deadline*" définie au niveau méta-modèle, est décrite au niveau modèle par un deuxième synchroniseur liant l'activité "Review" du modèle de procédé à la deadline "ReviewDeadline" dans le modèle d'exception.

Ce deuxième synchroniseur sera déclenché au changement d'état de l'activité Review lors de son démarrage et de sa terminaison (Code 13). Au démarrage de l'activité, le synchroniseur se chargera de créer la deadline correspondante à cette activité. Cette deadline contiendra la date de fin à laquelle l'activité devrait être terminée et qui ne sera précisée qu'au moment de l'exécution. A la terminaison de l'activité Review il n'y aura plus besoin de la deadline. A ce moment là, le synchroniseur va être sollicité pour la supprimer.
Le premier synchroniseur qui traduit la relation "*exception – activité*", sera quant à lui déclenché à la création de l'alarme "ReviewAlarme". Il transmettra à l'activité correspondante "Review" le traitement prévu par l'alarme pour cette activité (suspension, arrêt...).

Pour agir sur les entités des deux modèles (le modèle principal, et le modèle d'extension), les synchroniseurs utilisent le moteur de la Machine Virtuelle de chacun de ces deux domaines. La classe "ExceptionManager" que nous avons construit, constitue le moteur de la Machine Virtuelle du domaine de l'extension. Pour utiliser cette classe dans les synchroniseurs, nous avons défini un service abstrait "ExceptionManagerService" représentant l'interface de cette classe. Nous pouvons alors déclarer ce service dans le synchroniseur et l'utiliser en appelant les fonctions qu'il offre (voir le bloc "components" dans le Code 13).

```
import apel.motor.model.*;
aspect ReviewDeadlineSynchroniser ( int newState ) of Activity {
  when instance.getName( ).equals("Review") ;
  components {
     exceptionManager : ExceptionManagerService; }
  body( JAVA ) {
     Activity activity = instance;
     String activityName = activity.getName();
     if ( activity.isActive() ){ exceptionManager.createDeadline(activityName); }
     else if ( activity.isTerminated() ){ exceptionManager.deleteDeadline(activityName);} } }
```

Code 13 : Le synchroniseur de la relation "Review Activity - Review Deadline"

Niveau instance

A l'exécution du logiciel (Figure 42) lorsque l'activité "Review" démarre, le synchroniseur d'extension correspondant "ReviewDeadlineSynchroniser" est déclenché. Il crée l'entité deadline correspondante à cette activité-là. La création de cette entité deadline implique le démarrage de l'adaptateur "RwDdlnAdaptator" (Code 14) du fait que cet adaptateur intercepte la méthode "createDeadline" du moteur de la Machine Virtuelle du domaine de l'extension "ExceptionManager" (Code 12). Cet adaptateur execute l'outil "DeadlineEditor" à travers son service (Code 15). L'éditeur "DeadlineEditor" demande au responsable du procédé de préciser la date de fin pour l'activité "Review". Lorsque cette date est précisée, l'outil "DeadlineEditor" appelle l'adaptateur "SaveDateRwDdlnAdaptator" (Code 16), interceptant la méthode "saveDeadlineDate" de cet outil, qui se charge de transmettre la valeur de la date précisée au moteur "ExceptionManager" de la Machine Virtuelle du domaine de l'extension. Celui-ci attribue cette valeur à l'instance de la classe "ReviewDeadline" dans la couche conceptuelle de l'extension.

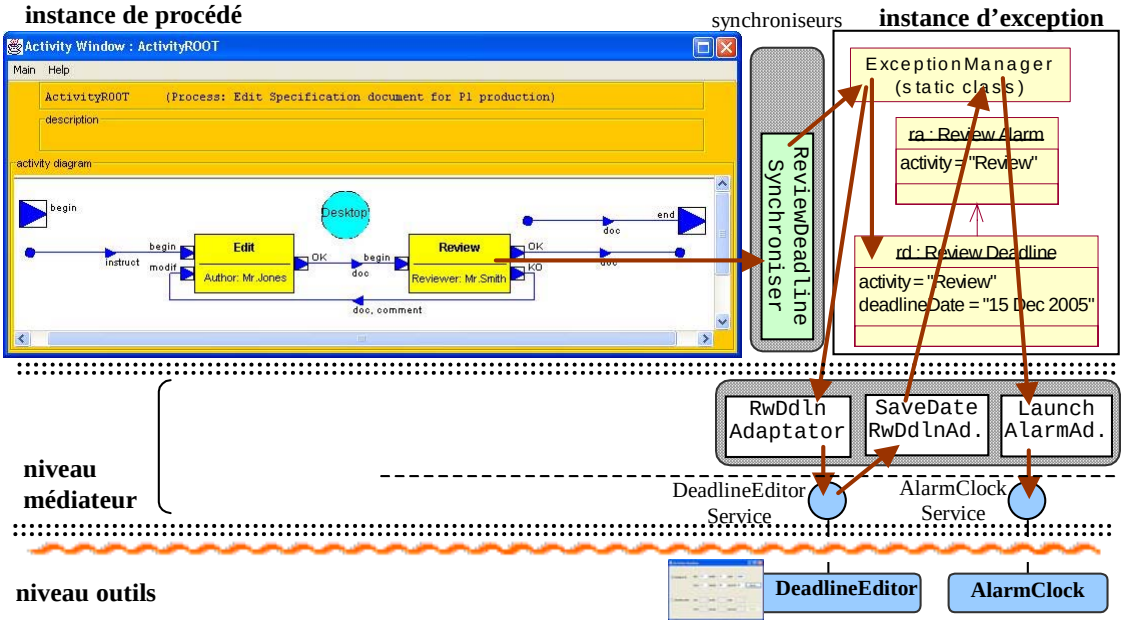


Figure 42 : Ajout du concept d'alarme et de deadline – l'exécution du logiciel

```
aspect RwDdlnAdaptator ( String activityName ) of ExceptionManager {
   components { deadlineEditor : DeadlineEditorService; }
   body( JAVA ) { deadlineEditor.launchEditor(activityName); } }
```

Code 14 : L'adaptateur "RwDdlnAdaptator"

```
public interface DeadlineEditorService {
   public void launchEditor(String activityName);
   public void saveDeadlineDate (String activityName, int year, int month, int day,
                      int hours, int minutes, int seconds); }
```

Code 15 : Le service "DeadlineEditorService"

```
aspect SaveDateRwDdlnAdaptator (String activityName, int year, int month, int day,
                      int hours, int minutes, int seconds) of DeadlineEditor {
   components { exceptionManager : ExceptionManagerService; }
   body( JAVA ) { exceptionManager.saveDeadlineDate ( activityName, day, month, year, hours,
                      minutes, seconds); } }
```

Code 16 : L'adaptateur "SaveDateRwDdlnAdaptator"

```
aspect LaunchAlarmAdaptor (String activityName, int year, int month, int day,
                      int hours, int minutes, int seconds) of ExceptionManager{
   components { alarmClock : AlarmClockService; }
   body( JAVA ) { alarmClock.alarmAtDate(activityName, year, month, day, hours,
                      minutes, seconds) ; } }
```

Code 17 : L'adaptateur "LaunchAlarmAdaptor"

```
public interface AlarmClockService {
  public void alarmAfterDuration (String activityName, int year, int month, int day,
                                     int hours, int minutes, int seconds);
  public void alarmAtDate (String activityName, int year, int month, int day,
                                     int hours, int minutes, int seconds) ;
  public void stopAlarm (String activityName);
  public void launchException (String activityName); }
```

Code 18 : Le service AlarmClockService

Lorsque la date limite pour l'activité Review est précisée et est attribuée à l'instance de "ReviewDeadline", l'outil "AlarmClock" est lancé à travers un adaptateur, interceptant la méthode "saveDeadlineDate" du moteur ExceptionManager (Code 17) et un service (Code 18). Cet outil correspond à une horloge vérifiant régulièrement si la date du deadline n'est pas dépassée.

Lorsque l'activité "Review" se termine, le synchroniseur "ReviewDeadlineSynchroniser" (Code 13) se déclenche, et supprime l'entité deadline correspondante à cette activité, de la couche conceptuelle de l'extension. Ceci car une fois que l'activité est terminée, il n'y a plus besoin de cette deadline. La suppression de la deadline va en conséquence arrêter l'outil "AlarmClock" (toujours à travers un adaptateur et un service).

Si la date limite est atteinte et que l'activité n'est toujours pas terminée, l'outil "AlarmClock" appelle la méthode "launchException" qui crée à travers un adaptateur une instance de la classe "ReviewAlarm" dans le domaine conceptuel de l'extension. A la création de cette instance, un synchroniseur et un adaptateur sont démarrés pour transmettre cette alarme à la couche conceptuelle de procédé d'une part et aux outils de l'extension se chargeant de réagir à cette alarme. Le synchroniseur démarré correspond à la relation *"alarme – activité"*. Il contient le traitement prévu par l'alarme pour cette activité. Dans notre exemple il suspend cette activité. L'adaptateur, démarré à la création de l'instance de l'alarme, lance un outil de

gestion d'exceptions qui s'occupe de l'exception levée (il avertit le responsable de l'application globale par exemple).

Les concepts d'exception que nous avons ajoutés font partie de la même extension. L'utilisateur final de l'application globale pourra choisir à chaque exécution s'il souhaite ou non utiliser cette extension. Il lui suffit de sélectionner dans une liste l'extension de méta-modèle qu'il souhaite utiliser, de la même façon à ce que nous avons expliqué pour l'extension sémantique.

4) L'utilisation d'un Editeur Graphique extensible

Dans notre travail de mise en place d'une extension de meta-modèle de procédé, nous avons construit un éditeur graphique extensible. Cela facilite l'édition des liens de synchronisation entre le modèle de procédé et le modèle d'exception. Cet éditeur peut être adapté à d'autres extensions de notre méta-modèle de procédé.

Cet éditeur graphique nous permet d'afficher un modèle de procédé construit à l'aide de l'éditeur APEL. Il permet également de définir des modèles (des "templates") de synchroniseurs liant les concepts du modèle de procédé aux concepts du modèle de l'extension. Nous pouvons également définir les éléments d'une liste contextuelle (sous forme de pop-up) qui s'affiche en cliquant sur un concept (une activité, un port...) du modèle de procédé (Figure 43).

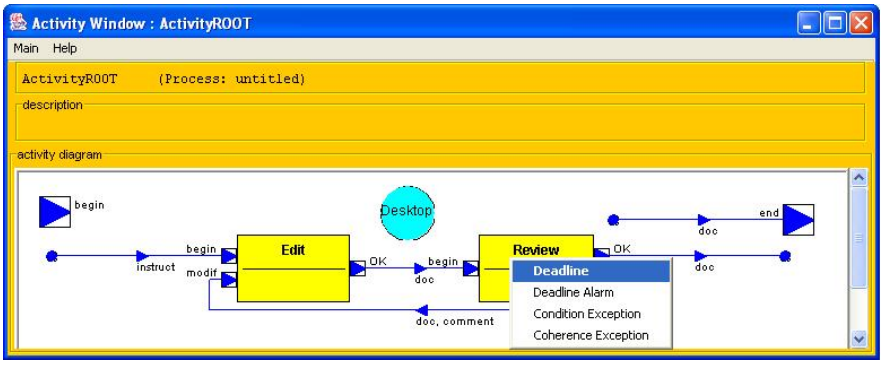


Figure 43 : Editeur graphique pour l'extension du méta-modèle de procédé

La Figure 43 illustre notre éditeur avec pour exemple le modèle de procédé de rédaction de document. Nous avons adapté cet éditeur pour l'extension du méta-modèle de procédé par le méta-modèle d'exception. Ainsi, en cliquant sur une activité (Review dans notre figure), la liste contextuelle s'affiche proposant les noms des différents types de concepts que nous pouvons lier à cette activité par un synchroniseur. Le choix d'un de ces types de concepts permet d'afficher le modèle du synchroniseur correspondant à ce concept. Ainsi, cet éditeur offre au concepteur de l'extension de méta-modèle une aide graphique pour la réalisation des synchroniseurs. Le concepteur n'aura plus qu'à remplir les valeurs précisant les détails de ces synchroniseurs.

Dans notre exemple (Figure 43), lorsque nous cliquons sur "Deadline" dans la liste contextuelle liée à l'activité Review, notre éditeur graphique crée le synchroniseur "ReviewDeadlineSynchroniser" (Code 13) dans le répertoire des synchroniseurs de l'extension de méta-modèle de notre logiciel. Pour cela, il utilise le modèle du synchroniseur de la relation *"Activity – Deadline"* (Code 19) en remplacent "<ActivityName> par le nom de l'activité "Review".

```
import apel.motor.model.*;
aspect <ActivityName>DeadlineSynchroniser ( int newState ) of Activity {
  when instance.getName( ).equals("<ActivityName>") ;
  components {
     exceptionManager : ExceptionManagerService; }
  body( JAVA ) {
     Activity activity = instance;
     String activityName = activity.getName();
     if ( activity.isActive() ){ exceptionManager.createDeadline(activityName); }
     else if ( activity.isTerminated() ){ exceptionManager.deleteDeadline(activityName);} } }
```

Code 19 : Le modèle du synchroniseur de la relation "Activity – Deadline"

2.2.4. Evaluation

Nous avons présenté à travers notre exemple les étapes nécessaires pour construire une extension de méta-modèle. Ces étapes peuvent être résumées ainsi :

- définir le méta-modèle de l'extension et construire la machine virtuelle qui l'implémente ;
- définir les liens de dépendance entre les concepts du méta-modèle de l'extension et les concepts du méta-modèle de notre logiciel ;
- spécifier les concepts que nous souhaitons ajouter au modèle de notre logiciel (spécifier le modèle d'extension) ;
- pour chaque lien de dépendance entre un concept du modèle de l'extension et un concept du modèle du logiciel, écrire le synchroniseur qui l'implémente ;
- construire les outils capables d'implémenter les nouveaux concepts et décrire le service (l'interface) de chacun de ces outils ;
- écrire dans FedeEditor l'adaptateur liant chacun des concepts de l'extension à l'outil (ou aux outils) qui l'implémente à travers son (leurs) service(s).

Nous remarquons que cela représente l'écriture de beaucoup de petits morceaux de code, à différents niveaux du logiciel, ainsi que l'implémentation de plusieurs outils, qui peuvent être plus ou moins complexes.

Néanmoins, la construction de l'extension de méta-modèle en suivant notre approche est relativement simple, par rapport à l'effort et le temps dépensé pour ajouter de nouveaux concepts à un logiciel (étendre son méta-modèle), en ajoutant directement du code dans ses fichiers sources. Nous pouvons expliques cela sur plusieurs points :

- En ce qui concerne la construction d'outils implémentant les nouvelles fonctionnalités, le temps dépensé est le même. Cependant, dans notre approche, il est possible d'utiliser des outils existant, et cela en spécifiant l'interface (le service de l'outil) affichant les méthodes de l'outil que nous pouvons utiliser ;
- En ce qui concerne la construction des synchroniseurs et des adaptateurs, nous avons vu que cela représente l'écriture de plusieurs petits morceaux de code, à deux niveaux différents du logiciel. Cependant, ces morceaux de code sont généralement de l'ordre de quelques lignes (moins de dix lignes). Ils sont écrits à l'aide d'un langage simple et extensible, s'adaptant au domaine de l'application. La gestion des synchroniseurs et des adaptateurs est faite à l'aide de l'éditeur graphique FedeEditor, qui offre beaucoup de facilités au constructeur du logiciel (vérification, compilation, organisation structurelle...) ;
- Et finalement, en ce qui concerne l'architecture du logiciel, notre approche nous permet la séparation de préoccupation. Ceci nous permet de passer facilement d'une extension à l'autre, et éventuellement de les combiner, afin d'exécuter notre logiciel avec ou sans cette (ou ces) extension. Ce changement de la manière d'utiliser le logiciel (tantôt avec et tantôt sans l'extension) est très difficile à mettre en place lorsque le logiciel ne contient pas une architecture adaptée.

Pour l'utilisateur du logiciel, l'utilisation de l'extension de méta-modèle est autant facile que l'utilisation de l'extension sémantique, car il suffit de sélectionner (ou désélectionner), dans FedeEditor, l'extension de méta-modèle qu'il souhaite utiliser (ou ne pas utiliser).

2.3. Raffinement

La troisième forme d'évolutivité que nous présentons dans ce chapitre est le raffinement.

2.3.1. Définition du raffinement

Comme nous l'avons précisé précédemment, nous nous basons dans notre approche sur un méta-modèle simple établi sur des éléments de base. Ces éléments de base sont de gros grain, et très généraux. Ils ne contiennent pas de détails. Le raffinement permet d'étendre ces éléments de base en leur ajoutant des informations complémentaires afin de les rendre plus spécifiques pour une application donnée. Ils peuvent ainsi répondre à la fonction que nous attendons d'eux dans le système que nous souhaitons construire.

2.3.2. Construction du raffinement

De la même manière que la construction d'une extension de méta-modèle, la construction du raffinement se fait en deux étapes : la construction de la Machine Virtuelle du Raffinement contenant le méta-modèle et le moteur de raffinement., puis la construction du raffinement de notre logiciel.

2.3.3. Exemple de raffinement : notre logiciel de rédaction de document

Nous allons reprendre notre exemple de rédaction de document pour détailler l'approche de raffinement de méta-modèle du logiciel.

1) Définition du méta-modèle de raffinement

Dans notre approche, la construction de logiciels dirigés par procédé se base sur un méta-modèle de procédé simple comprenant sept éléments de base : activité, produit, type de produit, port, rôle, ressource, et flots de données (Figure 21, Chapitre IV.). Ces sept éléments sont très généraux et ne contiennent pas d'attributs spécifiques.

Dans les logiciels dirigés par procédé que nous construisons, nous pouvons avoir besoin de plus de précisions sur certains de ces éléments. Par exemple, dans notre logiciel de rédaction de document, nous pouvons avoir besoin de détailler le concept de type de produit. Un type de produit peut être défini comme étant composé d'un responsable de produit, d'une version de produit, d'attributs, de droits et de fichiers (Figure 44). Nous proposons donc de raffiner le méta-modèle de procédé afin d'ajouter les informations complémentaires au concept de type de produit.

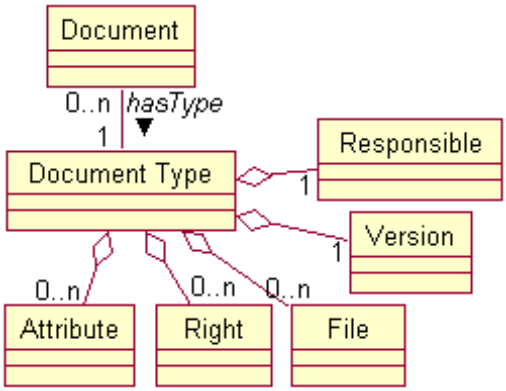


Figure 44 : Le méta-modèle représentant la structure détaillée du concept de type de produit

2) Les relations entre le domaine principal et le raffinement

Après avoir construit le méta-modèle de raffinement, nous définissons les relations entre le domaine principal (domaine de procédés) et le domaine de raffinement (domaine de documents dans notre exemple). Nous présentons dans cette section ces relations de raffinement pour les trois niveaux : méta-modèle, modèle et instance.

Niveau méta-modèle

Pour ajouter une structure de type de produit plus détaillée au méta-modèle de procédé nous définissons un méta-modèle contenant cette nouvelle structure de type de produit. Dans ce méta-modèle, le concept principal se nomme "Document". Il est composé des concepts : "Responsible", "Version", "Attribute", "Right", et "File" (Figure 44). Nous lions le concept de "Product Type" du méta-modèle de procédé au concept de "Document Type" du méta-modèle

ajouté par un lien de raffinement orienté de "Document Type" vers "Product Type". Ceci exprime qu'au niveau du méta-modèle, ce nouveau concept est le raffinement du concept de type de produit du modèle de procédé.
La relation de raffinement est une relation de multiplicité 1-1 qui indique qu'au niveau modèle chaque "Document Type" du modèle de raffinement correspond à un "Product Type" du modèle de procédé. Ainsi, dans notre exemple de rédaction de document, un type de produit du domaine de procédés devient désormais un type de document du domaine de raffinement.

Niveau modèle

Après avoir défini le méta-modèle de raffinement, nous définissons le modèle de documents pour notre logiciel de rédaction. Dans le modèle de procédé de ce logiciel nous avons défini trois types de produits : le type doc "docType", le type instruction "instructType" et le type commentaire "commentType" (Figure 45, modèle de procédé dans APEL editor). Pour chacun de ces types de produits, nous définissons le type de document correspondant. Ces trois types de document formeront alors le modèle de documents de notre logiciel de rédaction.

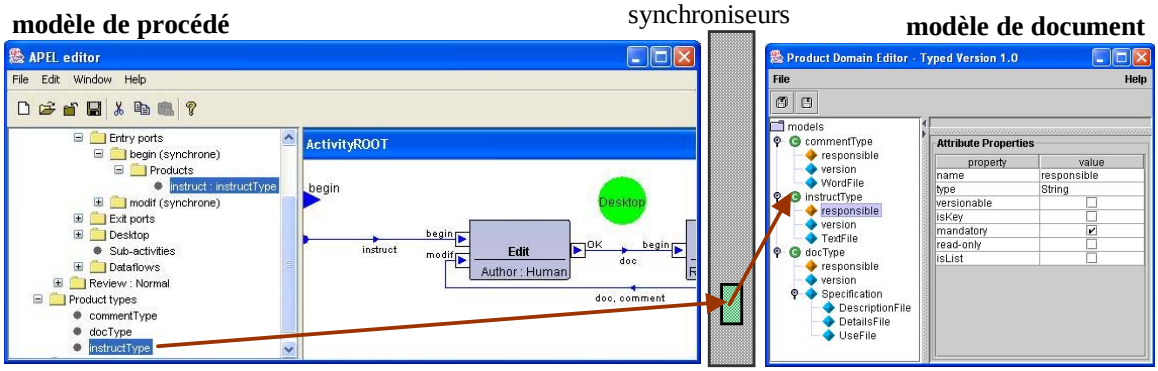


Figure 45 : le niveau modèle

Nous avons construit un éditeur permettant de définir le modèle de documents. Nous utilisons cet éditeur pour spécifier le modèle des trois types de documents docType, instructType et commentType conformément au méta-modèle de document défini précédemment. Nous choisissons que commentType contienne un responsable, un numéro de version et un fichier "WordFile", que instructType contienne un responsable, un numéro de version et un fichier "TextFile", et que docType contienne un responsable, un numéro de version, et trois fichiers "DescriptionFile","DetailsFile" et "UseFile" (Figure 45, modèle de document).

La relation de raffinement que nous avons défini au niveau méta-modèle est, comme nous l'avons mentionné précédemment, une relation de multiplicité 1-1. Ceci indique qu'un type de produit (instructType par exemple) du modèle de procédé correspond à un type de document (instructType) qui le raffine dans le modèle de document, et inversement. Nous déduisons de cette relation qu'à un produit "instruct" de type "instructType" du modèle de procédé correspond un document "instruct" de type "instructType" du modèle de document qui raffine ce produit. Cette relation se traduit alors au niveau modèle par des synchroniseurs de

raffinement, permettent de lier chaque produit au document correspondant qui le raffine. Ces synchroniseurs sont définis au niveau modèle et déclenchés au niveau instance (à l'exécution de notre logiciel). Nous décrivons ces synchroniseurs dans la section suivante en présentant leur exécution.

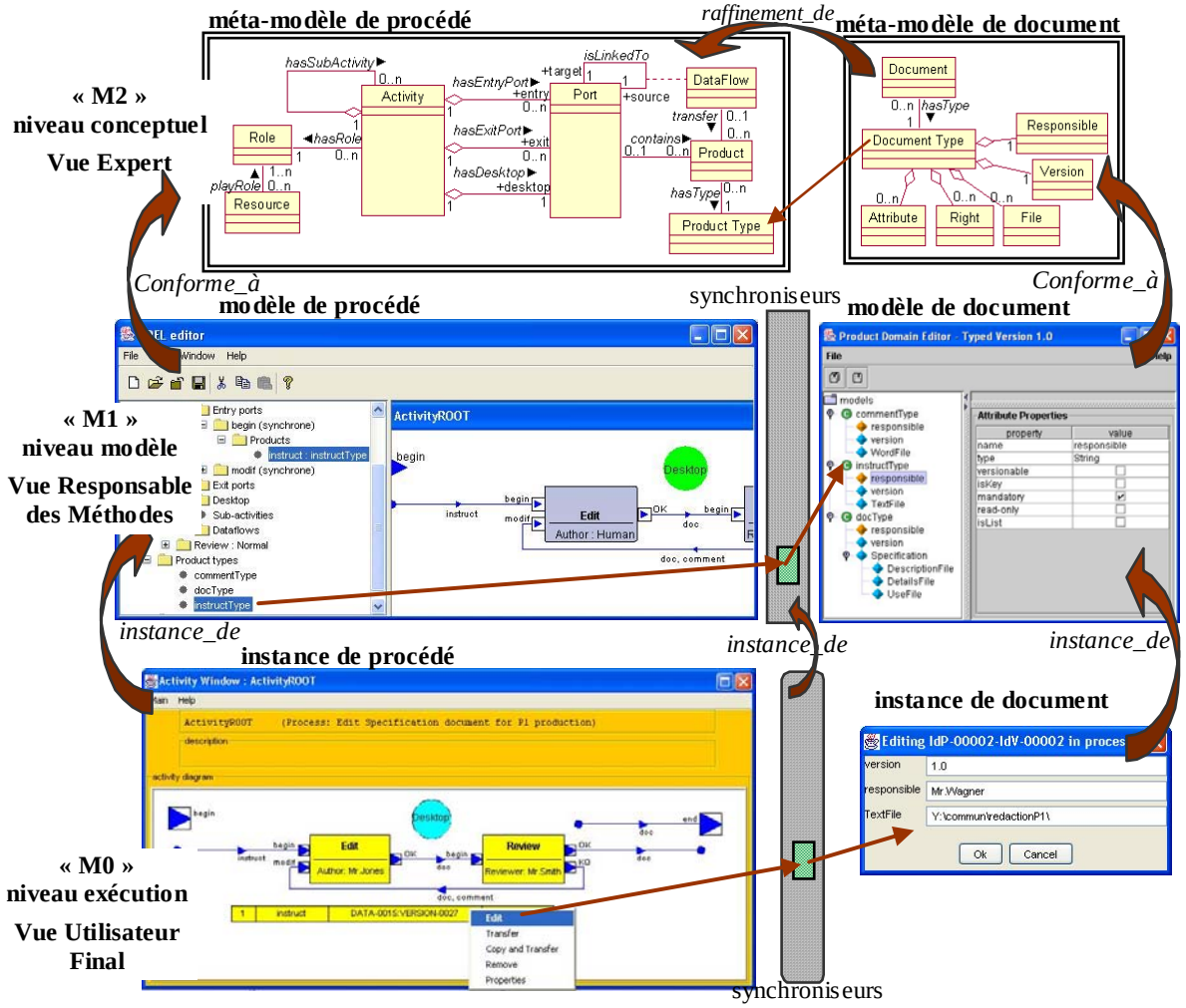


Figure 46 : Raffinement du concept de produit du méta-modèle

Niveau instance

Pour lier chaque produit (du domaine de procédés) au document correspondant (du domaine de documents), nous avons défini des synchroniseurs de raffinement au niveau modèle. Ces synchroniseurs sont déclenchés à l'exécution à chaque fois qu'un nouveau produit (ou une nouvelle version de produit) apparaît dans l'instance de procédé. Ceci arrive lorsqu'une activité crée une instance de produit afin de l'envoyer vers une autre activité ou vers la sortie du procédé. Ceci arrive également lorsque le procédé, au moment de son démarrage, crée une instance de produit afin de l'envoyer vers la première activité de ce procédé.

Lorsqu'un ce synchroniseur de raffinement est déclenché, il crée dans le domaine de documents une instance de document correspondante (de type correspondant) à cette nouvelle

instance de produit apparue dans le domaine de procédés. Ce synchroniseur insère également un attribut dans l'instance de produit indiquant la référence vers l'instance de document correspondante (qui la raffine) et qui correspond à la relation de raffinement.

La création de l'instance de document déclenche, dans le domaine de documents, un adaptateur ouvrant un outil d'édition de document que nous avons construit. Cet outil d'édition s'adapte au type du document créé afin de permettre à l'utilisateur de l'éditer. Les valeurs inscrites dans cet éditeurs sont répercutées à l'instance de document dans la couche conceptuelle du domaine de documents à travers un deuxième adaptateur (Figure 47).

L'outil d'édition sera ouvert chez le responsable de l'activité, que nous connaissons car le synchroniseur nous a transmis la valeur du responsable de l'activité du procédé contenant ce produit.

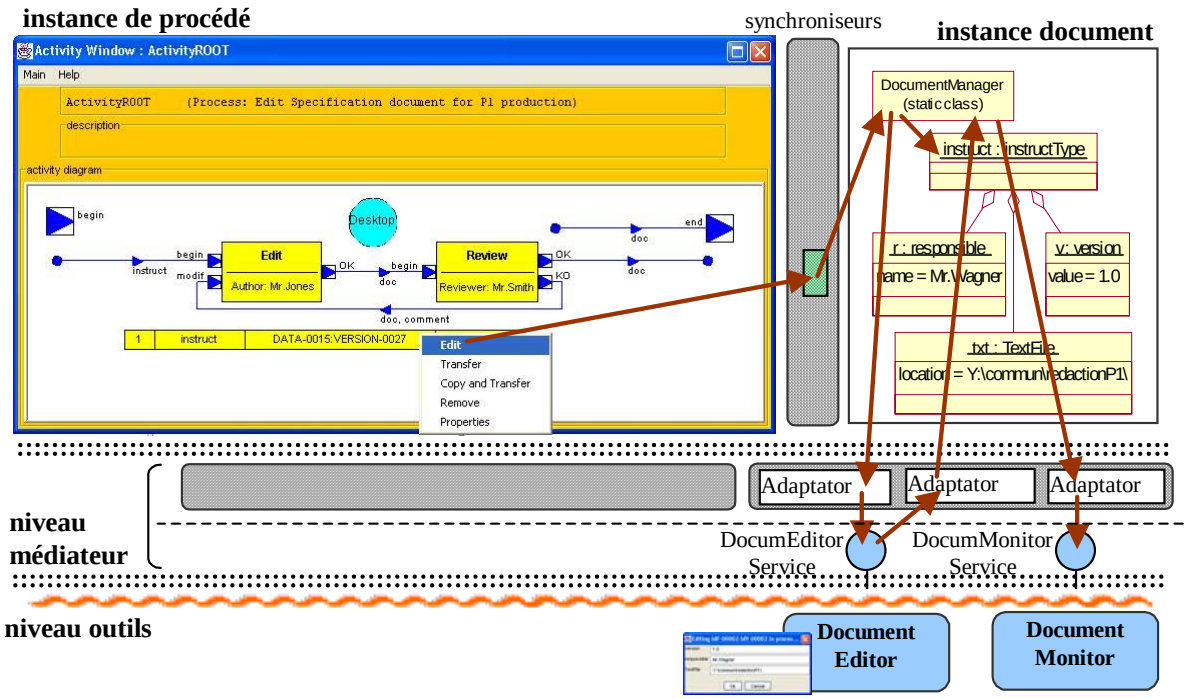


Figure 47 : Raffinement du concept de produit – l'exécution du logiciel

Ces instances de documents créées peuvent être manipulés comme nous le souhaitons. Nous pouvons par exemple choisir que les détails de chacune de ces instances de document soient affichés dans une application de supervision. Ainsi, à chaque fois qu'une nouvelle instance de produit apparaît dans le procédé, elle sera affichée dans cette application de supervision. Ceci nous permettra de consulter les valeurs de la structure raffinée de chaque produit, chose que nous ne pouvons pas faire uniquement avec l'application de supervision de l'exécution du procédé. Pour cela, nous avons implémenté un outil de supervision "Document Monitor" permettant d'afficher des entités conformes aux modèles de documents de notre raffinement. Ainsi, dès que de l'instance de document reçoit sa valeur détaillée (raffinée), cette valeur sera

transmise (à travers un adaptateur et un service) à l'outil "Document Monitor", afin que cet outil puisse l'afficher (Figure 47).

2.3.4. Evaluation

Nous voyons, à travers cet exemple, que les étapes nécessaires pour construire un raffinement sont similaires à ceux de la construction d'extension de méta-modèle. Ceci avec la même complexité, dont nous avons parlé dans la section 2.2.4, et qui est relativement faible comparant au travail de raffinement d'un concept du logiciel, en ajoutant directement du code dans ses fichiers sources.

La spécificité des relations de raffinement est le fait que ce sont des relations 1-1. Il y a donc une correspondance directe entre un concept du méta-modèle du logiciel est un concept détaillé qui le raffine. Cette relation est concrétisée par l'insertion d'un attribut dans l'instance du concept de base correspondant à la référence vers l'instance qui la raffine. Cette insertion est faite pendant l'exécution à l'aide de nos synchroniseurs. Cela facilite la gestion des relations entre le concept de base et son raffinement et simplifie l'écriture des synchroniseurs du fait qu'ils peuvent utiliser la référence directe vers l'instance raffinée.

Il est important de noter que les différents niveaux, que nous avons présentés dans la Figure 46 (ainsi que dans la section précédente, Figure 41), sont gérés par des personnes ayant des profils différents :

- dans le niveau conceptuel (niveau méta-modèle) nous avons la vue "Expert" ;
- dans le niveau modèle nous avons la vue "Responsable des Méthodes" ;
- dans le niveau exécution (niveau instance) nous avons la vue "Utilisateur Final".

L'expert définis le méta-modèle du raffinement et les liens entre ce méta-modèle et le méta-modèle du logiciel. Ce travail demande une forte connaissance du méta-modèle du logiciel de base et en modélisation. Il peut se charger également de construire la machine virtuelle du raffinement implémentant ce méta-modèle de raffinement.

Le responsable des méthodes connaît bien le modèle du logiciel. Il définit le modèle de raffinement qui sera utilisé. Il construit les synchroniseurs, les adaptateurs, les services et les outils implémentant les liens entre le modèle de raffinement et le modèle du logiciel. Nous avons parlé du niveau de complexité de ce travail au début de cette section.

Pour l'utilisateur final, l'utilisation du logiciel raffiné est assez simple car il n'a qu'à choisir dans une liste de FedeEditor la ou les extensions qu'il souhaite utiliser.

2.4. Composition

La quatrième forme d'évolutivité que nous présentons dans ce chapitre est la composition.

2.4.1. Définition de la composition

L'architecture à trois couches que nous proposons nous permet de construire des logiciels de différents domaines métier. Comme nous l'avons indiqué, cette architecture apporte beaucoup d'avantages du fait qu'elle sépare la logique du logiciel de son implémentation et qu'elle utilise un méta-modèle métier minimal établi sur des éléments de base pour décrire cette logique métier du logiciel. Chacun de ces logiciels est alors spécialisé dans un domaine métier qu'il maîtrise.

Pour construire des logiciels rassemblant le savoir-faire de plusieurs domaines de compétence, nous avons besoin de composer ces logiciels spécialisés afin de les faire interopérer. Nous proposons alors une quatrième forme d'évolutivité : la composition qui permet de faire interopérer des logiciels de différents domaines métier en liants leurs concepts au niveau méta-modèle.

2.4.2. Construction de la composition

Pour construire une composition de logiciels en utilisant notre approche, il est important que les concepts représentant la logique de ces logiciels soient séparés de leur implémentation, et réifies à l'exécution. Et ceci car notre composition consiste à lier les concepts représentant la logique de ces domaines, au niveau méta-modèle de ces logiciels. Ceci est la seule condition pour construire une composition entre ces logiciels. Ainsi, il est possible d'utiliser notre approche pour construire une composition de logiciels, que ces logiciels soient basés sur notre architecture à trois couches ou pas, du moment ou les concepts décrivant la logique de ces logiciels sont réifiés, et contenus dans un noyau séparé de l'implémentation.

2.4.3. Exemple de la composition : notre logiciel de rédaction de document

Dans notre exemple, nous souhaitons associer notre logiciel de rédaction de document avec un logiciel de gestion d'espace de travail (workspace) pour pouvoir gérer l'espace de travail des personnes qui participeront à l'édition et à la révision de document.

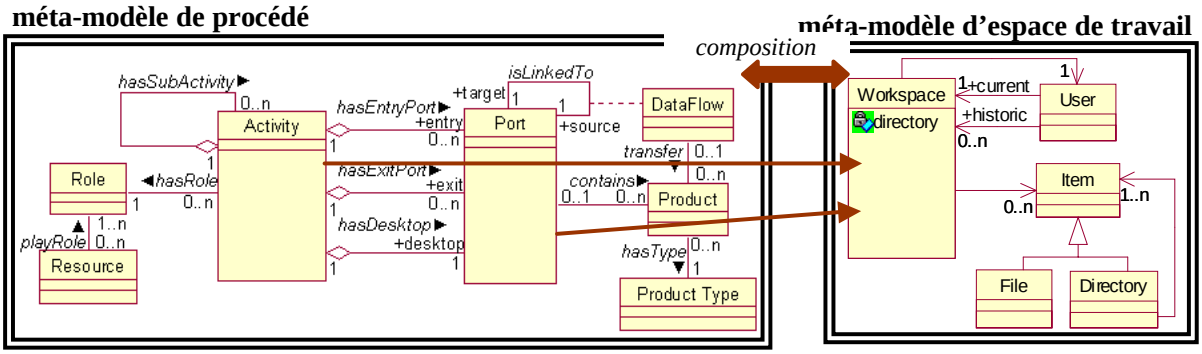


Figure 48 : Composition du méta-modèle de procédé et du méta-modèle d'espace de travail (workspace)

Comme pour les autres extensions, nous analysons cette relation de composition entre le domaine de procédés et le domaine d'espaces de travail selon les trois niveaux : méta-modèle, modèle et instance.

Niveau méta-modèle

Le but est de composer les deux méta-modèles procédé et espace de travail (Figure 48). Pour cela, nous créons un lien de composition entre le concept "Activity" du méta-modèle de procédé et le concept "Workspace" du méta-modèle d'espace de travail ainsi qu'un lien de composition entre le concept "Port" du méta-modèle de procédé et le concept "Workspace" du méta-modèle d'espace de travail.

Le lien *"Activity – Workspace"* a pour but de synchroniser l'espace de travail par rapport à l'activité : un espace de travail sera crée (ou détruit) lorsque l'activité correspondante est crée (ou détruite). Le deuxième lien *"Port – Workspace"* à pour butde synchroniser les produits arrivant dans le port "Desktop" d'une activité du procédé avec les produits contenus dans l'espace de travail du responsable de cette activité. A chaque fois qu'un produit arrive dans le port "Desktop" d'une activité, il faut ajouter le document correspondant à ce produit dans l'espace de travail du responsable de l'activité. Et à chaque fois qu'un produit part du port de sortie d'une activité, il faut retirer le document correspondant à ce produit de l'espace de travail du responsable de l'activité.

Ces deux liens sont orientés du méta-modèle de procédé vers le méta-modèle d'espace de travail car dans notre exemple c'est le procédé qui va influencer le domaine d'espaces de travail.

Niveau modèle et niveau instance

Pour traduire le lien "A*ctivity – Workspace*", nous définissons au niveau modèle un synchroniseur de composition qui se déclenchera au démarrage et à la terminaison de chaque activité du procédé (Figure 49). Ce synchroniseur a pour but, lors du démarrage de l'activité (respectivement de la terminaison), de créer (respectivement de supprimer) dans le domaine d'espaces de travail l'entité "Workspace" correspondante à cette activité. La création et la suppression de l'entité "Workspace" déclenche des adaptateurs qui se chargent d'appeler l'outil de gestion d'espace de travail à travers son service abstrait. Cet outil effectue la création ou la suppression d'un répertoire espace de travail sur la machine du responsable de l'activité.

Pour traduire le lien "*Port – Workspace*", nous définissons au niveau modèle deux synchroniseurs de composition S1 et S2. Le premier synchroniseur se déclenche à la sortie d'un produit du port de sortie d'une activité et le deuxième se déclenche à l'arrivé du produit dans le port "Desktop" d'une activité.

Lorsqu'un produit sort du port de sortie d'une activité, le synchroniseur S1 se déclenche (Figure 49). Il transmet l'information concernant le produit sortant de l'activité du domaine de

procédés à l'entité "Workspace" du responsable de cette activité dans le domaine d'espaces de travail. Ainsi, l'entité "Workspace" du responsable de cette activité s'occupera de transférer le document de l'espace de travail correspondant à ce produit vers une base de données contenue dans un serveur central et aussi de l'effacer de l'espace de travail du responsable de l'activité. Pour cela, il fait appel aux outils gestionnaire d'espace de travail et transfert de fichiers correspondant à travers des adaptateurs et des services.

Lorsqu'un produit arrive dans le port "Desktop" d'une activité, le synchroniseur S2 se déclenche. De la même manière que dans S1, il transmet l'information concernant le produit arrivant au port "Desktop" de l'activité du domaine de procédés à l'entité "Workspace" du responsable de cette activité dans le domaine d'espaces de travail. Ainsi, l'entité "Workspace" du responsable de cette activité s'occupera de transférer le document correspondant à ce produit de la base de données du serveur central vers l'espace de travail du responsable de l'activité. comme dans le cas précédant, il fait appel aux outils gestionnaire d'espace de travail et de transfert de fichiers correspondants, à travers des adaptateurs et des services.

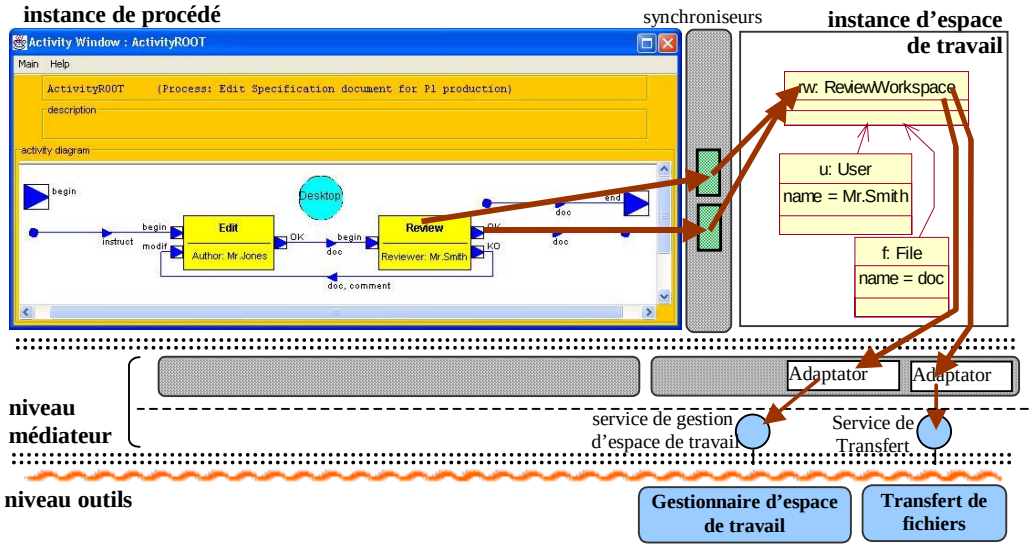


Figure 49 : Composition du procédé et de l'espace de travail (workspace) - l'exécution du logiciel

2.4.4. Evaluation

La composition de logiciels nous permet de cumuler les compétences et le savoir-faire dans plusieurs domaines métier. Elle utilise la même technique de synchroniseurs pour lier les concepts des deux méta-modèles de la composition et la même technique d'adaptateurs, de services et d'outils, pour implémenter ces concepts. Le niveau de complexité de cette construction est donc similaire à celui de l'extension de méta-modèle et du raffinement (sections 2.2.4 et 2.3.4) qui n'est pas très élevé.

La composition est aussi souvent utilisée pour lier notre logiciel avec un logiciel d'un autre domaine construit précédemment. Pour réaliser cela, il est important que les concepts représentant la logique de ce logiciel soient séparés de leur implémentation et réifiés à

l'exécution. Notre composition consiste alors à lier, via des synchroniseurs, ces concepts aux concepts du méta-modèle de notre logiciel. Ce travail peut être plus ou moins complexe, car il dépend de la connaissance que l'on a de la sémantique des concepts du nouveau logiciel. Il est cependant largement plus simple que la reconstruction complète des fonctions de ce nouveau logiciel. Il nous offre également l'avantage de pouvoir à tout moment exécuter ces deux logiciels séparément ou de façon commune, en le précisant simplement dans notre éditeur graphique FedeEditor.

3. Application à l'orchestration de services web

Nous avons présenté, dans le chapitre III, plusieurs langages d'orchestration et de chorégraphie de services web. Chacun de ces langages a apporté de nouveaux concepts et a redéfini certains concepts déjà connus par les autres langages. Ceci a créé une multitude de concepts se chevauchant parfois, ainsi qu'une multitude de façons de les manipuler. Aujourd'hui, les travaux autour de l'orchestration de services web sont toujours en cours de définition. De nouveaux langages contenant de nouveaux concepts vont encore apparaître en proposant de nouvelles idées et de nouvelles façons de manipuler ces concepts.

Dans cette thèse, nous avons présenté notre proposition pour la construction de logiciels de coordination d'outils via un procédé. Ces logiciels ont une architecture à trois couches : conceptuelle, médiateur et outils. La couche conceptuelle contient le procédé de coordination décrivant la logique de notre logiciel. Ce procédé est basé sur un méta-modèle de procédé simple, contenant un minimum de concepts. Ceci facilite la compréhension et la manipulation de ces concepts de base pour le concepteur du logiciel de coordination et évite de surcharger le méta-modèle de procédé par des concepts rarement utilisés.

Cependant, lorsque nous construisons des applications d'orchestration, basées sur un procédé conforme à ce méta-modèle minimal, il nous arrive d'avoir besoin d'un ou de plusieurs concepts additionnels ou plus détaillés que ceux du méta-modèle minimal. Ce besoin peut aussi arriver lors de l'évolution de notre application d'orchestration. Nous devons donc permettre cette extension de concepts, tout en gardant la simplicité du méta-modèle, la facilité de manipulation des concepts ainsi que la facilité de construction du logiciel utilisant ces nouveaux concepts.

Nous avons présenté dans ce chapitre, notre approche qui nous permet d'étendre et de faire évoluer les applications construites à l'aide de notre plate-forme Mélusine. Nous avons expliqué les quatre formes d'évolutivité que nous avons défini et qui nous permettent d'étendre facilement notre application. Ceci est réalisé en étendant le modèle de l'application du niveau conceptuel. Cette approche nous permet de modifier sémantiquement les concepts du modèle de l'application, d'ajouter de nouveaux concepts, de raffiner les concepts existants et de composer notre application avec une autre application afin d'utiliser ses fonctionnalités. Elle permet cela tout en séparant le modèle de notre application du modèle de l'extension ce qui permet de construire des modèles basés sur des concepts simples et faciles à manipuler. Cela permet également de choisir d'exécuter notre application avec ou sans l'extension que

nous avons réalisée. Nous avons le choix au moment de l'exécution d'ajouter ou pas ces nouveaux concepts à notre application.

Notre approche peut être autant appliquée à l'orchestration de services web, qu'aux logiciels de coordination d'outils via un procédé, ainsi qu'à des applications construites en respectant l'architecture proposée par notre plate-forme Mélusine.

4. Synthèse et Avantages de notre approche

Notre approche, présenté dans ce chapitre, permet d'étendre et de faire évoluer les logiciels construits à l'aide de notre plate-forme Mélusine. Nous avons défini, dans notre approche quatre formes d'évolutivité :

- Extension sémantique : permet de modifier la sémantique d'un concept du méta-modèle ;
- Extension de méta-modèle : consiste à ajouter de nouveaux concepts au méta-modèle ;
- Raffinement : consiste à étendre un (ou plusieurs) élément(s) de base, pour que cet élément puisse contenir tous les détails qui le concernent ;
- Composition : permet de faire interopérer plusieurs environnements en liant les concepts du niveau méta-modèle de ces environnements.

Ces quatre formes d'évolutivité peuvent être combinées selon le besoin.

Dans ces quatre formes, l'évolutivité du logiciel est toujours définie séparément de notre logiciel afin de ne pas mélanger les nouveaux concepts spécialisés (ajoutés pour un besoin particulier) avec les concepts de base de notre méta-modèle minimal. Cette séparation des concepts de base et des concepts spécifiques facilite la compréhension et la manipulation, car le concepteur du logiciel ne manipule qu'un nombre minimal de concepts séparés par domaine d'application.

L'implémentation de ces nouveaux concepts est également séparée, dans l'architecture que nous proposons, de l'implémentation de notre logiciel. Ce qui permet de bien structurer notre application globale.

Les nouveaux concepts sont liés aux concepts de notre application par des synchroniseurs. Les synchroniseurs peuvent être activés au moment de l'exécution, ce qui permet d'exécuter notre logiciel avec son évolution, ou désactivée, au cas où le logiciel s'exécuterait dans son état d'origine avant de définir son évolution. Il est important de noter que les quatre formes d'évolutivité proposés ne modifient pas le logiciel que nous souhaitons faire évoluer.

L'architecture que nous proposons pour l'évolution des logiciels est une architecture modulaire du fait de la séparation entre les différents domaines d'application et d'extension. Cette modularité offre beaucoup de flexibilité, simplifie la maintenance et facilite la réutilisation.

Notre approche proposant ces quatre formes d'évolutivité s'adresse à un très large spectre d'applications, en particulier les applications couvrant différents domaines d'application. Elle est applicable à tout logiciel, même si l'architecture de ce logiciel n'est pas tout à fait conforme à l'architecture de notre plate-forme Mélusine. La seule condition pour pouvoir appliquer nos quatre formes d'évolutivité à un logiciel est que les concepts représentant la logique de ce logiciel soient séparés de leur implémentation, et réifies à l'exécution.

Chapitre VI.

Conclusions et perspectives

1. Synthèse des travaux effectués

L'objectif de cette thèse est d'étudier les problèmes de la coordination d'applications à l'aide d'un procédé et de trouver des solutions à ces problèmes. Pour cela, nous avons étudié les différents domaines utilisant des procédés. Nous avons identifié les caractéristiques des divers types de procédé ainsi que leurs avantages et inconvénients.

Cette étude nous a permis d'identifier les besoins des procédés de coordination que nous résumons par les points suivants :

- Le besoin d'un procédé flexible, facile à modifier et à faire évoluer ;
- Le besoin de couplage faible entre le procédé et les applications qu'il coordonne ;
- Le besoin d'un formalisme de haut niveau décrivant la logique de l'application ;
- Le besoin de permettre la coordination d'applications utilisant diverses technologies (services web, programmes locaux ou distants, composants, COTS, etc.) ;
- Le besoin de simplifier les concepts du procédé et de se baser sur un minimum de concepts afin de ne pas surcharger le modèle de procédé par des concepts rarement utilisés ;
- Le besoin de permettre l'évolution et l'introduction de nouveaux concepts afin d'ajouter à ce modèle de procédé minimal les concepts métier spécialisés dont nous avons besoin pour notre application de coordination.

Nous avons aussi identifié les besoins concernant le système de gestion de coordination :

- Le besoin d'un environnement performant, gérant l'interopérabilité et la reprise sur pannes et contenant des outils d'aide pour l'utilisateur (outils d'analyse, de test, et de débogage) ;
- Le besoin de construire des logiciels spécialisés, s'occupant de l'ensemble de nos besoins, afin de réaliser une gestion de coordination cohérente et unifiée.

Pour répondre à l'ensemble de ces besoins, nous proposons dans cette thèse notre environnement Mélusine qui permet de construire des logiciels ayant une architecture à trois couches : concepts, médiateur, et outils. L'approche que nous utilisons dans cet environnement consiste à se baser, au niveau conceptuel, sur un meta-modèle minimal ne contenant que les concepts de base. Ce meta-modèle minimal peut être étendu afin de faire évoluer cet environnement. Nous définissons quatre classes d'évolutivité permettant d'étendre

le logiciel construit à l'aide de notre environnement : l'extension sémantique, l'extension de méta-modèle, le raffinement, et la composition. Ces quatre formes d'évolutivité peuvent être combinées selon les besoins.

2. Leçons retenues

Il est important, de séparer la logique de l'application de son implémentation, et de ne pas lier ces deux parties de l'application par des liens rigides. Et cela pour permettre la flexibilité de substitution d'un outil par un autre, et la flexibilité de modification de la logique de l'application. Pour cela, nous proposons une architecture à trois couches : concepts, médiateur, et outils. La couche conceptuelle contient la logique de l'application, elle contient le procédé dans les logiciels de coordination.

Il est important également de se baser sur un meta-modèle conceptuel minimal, ne contenant que les concepts de base. Cela permet de simplifier la compréhension des concepts et de faciliter leur manipulation en ne surchargeant pas le meta-modèle par des concepts spécifiques et rarement utilisés. Pour cela, dans notre environnement Mélusine, la couche conceptuelle se base sur un meta-modèle minimal, qui est pour les logiciels de coordination d'applications par procédé le meta-modèle APEL.

Il est essentiel de ne pas mélanger les nouveaux concepts ajoutés par extension, raffinement ou composition, avec les concepts de base de notre méta-modèle minimal. Ceci pour garder cette caractéristique de ne manipuler qu'un nombre minimal de concepts, séparés par domaine d'application. Et aussi pour ne pas modifier le logiciel que nous souhaitons faire évoluer, ainsi nous pourrons exécuter notre logiciel dans son état d'origine ou avec certaines ou toutes ses extensions.

3. Mes contributions

Ce travail s'inscrit dans les travaux de notre équipe autour de la plate-forme Mélusine et des "Fédérations". Mes contribution peuvent être résumées dans les points suivants :

- La participation à la construction de la plate-forme Mélusine : la plate-forme Mélusine est le résultat du travail d'un groupe de plusieurs personnes dans notre équipe. J'ai fait partie de ce groupe et j'ai ainsi participé à la mise en place de cette plate-forme ;

- La réalisation de la Démo Métier du projet Centr'Actoll, qui est une implémentation industrielle construite à l'aide de Mélusine : dans cette Démo Métier, j'ai utilisé Mélusine pour étendre un logiciel patrimonial industriel construit sur des anciennes technologies afin de l'enrichir par de nouvelles fonctionnalités se basant sur des technologies très récentes ;

- L'introduction de l'orchestration de services web parmi les usages de notre environnement de procédé APEL. Et cela en utilisant la plate-forme Mélusine pour la construction des applications d'orchestration ;

- L'étude des différents domaines utilisant des procédés et l'identification des besoins des logiciels de coordination par procédé ;

- L'introduction du concept d'extension de méta-modèle dans notre plate-forme Mélusine ;

- La classification des différentes formes d'évolutivité en quatre classes : l'extension sémantique, l'extension de méta-modèle, le raffinement et la composition.

Ce travail a donné lieu à cinq publications dont un article publié dans un Journal International [ES 05c], deux articles publiés dans des conférences internationales [ES 05a, ES 05b], un article publié dans un Workshop européen [EVLSV 03], et un article publié dans une revue francophone [SE 05].

4. Perspectives

Nous avons identifié plusieurs perspectives suite à ces travaux de thèse :

- **Extensions non-fonctionnelles**. Etudier l'extension de méta-modèle d'un logiciel par des concepts non-fonctionnels tels que les transactions. Nous avons identifié cette idée au cours de notre travail sur l'extension de méta-modèle que nous avons présenté dans la section 2.2 du Chapitre V. Cependant, elle n'a pas encore pu être approfondie. Cette fonctionnalité est très intéressante notamment pour les applications d'orchestration de services web. Ainsi, en formant une extension de méta-modèle pour exprimer les transactions du logiciel, nous serons capables de choisir d'exécuter notre logiciel avec cette propriété non-fonctionnelle ou sans elle, et ceci suivant certaines caractéristiques, comme par exemple le coût de cette propriété par rapport aux performances attendues de notre logiciel ;

- **Domaines non fonctionnels**. En poursuivant l'idée ci-dessus, s'il s'avère possible que les domaine non fonctionnels soient autonomes il est alors possible de doter n'importe quel domaine de propriétés non fonctionnelles, par simple composition. Par exemple, la propriété transactionnelle pourrait être simplement tissée avec tout autre domaine, sans avoir a modifier le code de ces domaines. Nous avons là une autre approche des conteneurs extensibles ;

- **Substitution dynamique d'outil** en cours d'exécution. Pour le moment, la substitution d'un outil par un autre est facilement réalisable tant que cet outil n'a pas commencé son

exécution. Il serait intéressant de couvrir également la substitution en cours d'exécution. Cela pourrait être réaliser de deux façons :

1) A travers l'ajout de certains mécanismes de compensation et de recherche dynamique d'outils, au niveau de la couche de médiation. Cette recherche dynamique pourrait être restreinte à la sélection d'un outil parmi plusieurs, choisis à l'avance dans notre logiciel. Elle pourrait aussi être élargie à de la découverte dynamique de services web sur le net. Ce sujet, concernant la découverte dynamique de services web a été démarré dans notre équipe dans un travail de Master Recherche [DLR 04]. Il serait intéressant d'étudier ce sujet de manière plus approfondie, en étudiant les ontologies et les critères de sélection de services web conformes aux besoins de notre application,

2) A travers une extension non-fonctionnelle, mais qui ne serait pas, cette fois-ci, pour étendre le meta-modèle. Cette extension se chargerait de la capture d'exceptions arrivant dans la couche médiateur, comme par exemple perdre la connexion à un outil, et elle sera responsable de la substitution de cet outil. Cela donne une nouvelle perspective : étudier l'extension non-fonctionnelle au niveau de la couche médiateur,

- **Généraliser la médiation** (proxy). Il faudrait approfondir notre travail sur les liens entre les services abstraits et les outils. Pour le moment, notre plate-forme Mélusine propose des facilités techniques pour lier un service abstrait avec un service web ou un outil distant (s'exécutant sur une machine différente de celle contenant le moteur de Mélusine). Cependant, un travail d'emboîtage de l'outil doit être réalisé manuellement par le concepteur du logiciel afin de rendre cet outil conforme au service abstrait défini dans l'application. Pour faciliter cette tâche des travaux doivent être menés autour de la correspondance entre les fonctionnalités offerte par un outil et celles requises par un service abstrait, ainsi que la conformité des données échangées entre l'outil et le service abstrait. Il s'agit d'une généralisation des travaux sur la médiation ;

- **Exécution distribuée et chorégraphie**. Il serait intéressant d'étudier la possibilité de distribuer l'exécution de notre logiciel de coordination. Notre plate-forme Mélusine permet la conception et l'exécution des logiciels de coordination de façon centralisée. Nous considérons que le procédé de coordination centralisé, regroupant la logique de l'application en entier, est très intéressant du fait que cela apporte une vision globale de l'application. Cependant, il serait intéressant de distribuer l'exécution du logiciel, afin d'optimiser les performances. Cela peut être réalisé en découpant le procédé en plusieurs parties exécutées sur différentes machines. Néanmoins, nous pensons qu'il serait nécessaire de garder un outil de supervision central gérant le bon déroulement de l'exécution. Cette distribution de l'exécution engendre plusieurs problématiques :

  1) La distribution du moteur d'exécution sur plusieurs machines est l'une de ces problématiques. Ce travail n'est pas simple, car il faut réussir à faire collaborer les diverses copies distribuées du moteur pour réaliser l'exécution et avoir des résultats

plus performants qu'une exécution centralisée. S'ajoutent à cela les questions de fiabilité du réseau, les questions de sécurité, la gestion distribuée de reprise sur pannes… ,

2) Définir les critères de découpage d'un procédé de coordination afin d'exécuter ces différentes parties sur plusieurs machines. Ces critères pourraient être liés à l'emplacement géographique des outils concernés par cette partie de la coordination et des responsables ayant des décisions à prendre dans cette coordination,

3) Etudier l'automatisation du découpage de ce procédé de coordination global à partir des critères de découpage définis pour l'exécution distribuée ainsi qu'étudier la représentation de ce procédé découpé (la vue globale/les vues partielles). Aujourd'hui, Mélusine nous permet de réaliser des applications de coordination où le procédé de coordination est structuré hiérarchiquement. Une possibilité serait de réaliser un procédé représentant la vue globale (gros grain) et où chaque tâche de ce procédé serait exécutée sur une des machines participantes. Cette tâche contiendra le procédé de coordination s'exécutant sur cette machine,

4) Etudier les différentes formes d'évolutivité que nous avons défini par rapport à l'exécution distribuée de notre application. Nous avons étudié l'évolutivité des logiciels construit à l'aide de notre plate-forme Mélusine en considérant que le niveau conceptuel de notre logiciel s'exécute de manière centralisée. Lors d'une exécution distribuée du niveau conceptuel de notre logiciel, l'extension doit être également distribuée. Il serait donc nécessaire d'étudier la meilleure façon de distribuer l'extension que nous définissons pour ce logiciel,

- **Composition hiérarchique et fédération de fédérations**. Mélusine permet de réaliser des applications de coordination où le procédé est structuré hiérarchiquement. Mais cette composition hiérarchique concerne la couche conceptuelle de notre application. Il serait intéressant d'utiliser un logiciel construit à l'aide de notre plate-forme Mélusine comme étant un outil dans une autre application construite avec Mélusine. Ceci peut être réalisé facilement du fait que Mélusine accepte toute sorte d'outil. Cependant, nous n'avons encore jamais réalisé ce genre d'application, car nous n'en avons pas encore eu le besoin. Ceci implique de pouvoir faire exécuter simultanément deux moteurs de Mélusine sur la même machine ;

- **Evolution dynamique**. Etudier l'évolutivité dynamique de nos logiciels construits avec Mélusine. Nous avons défini quatre formes d'évolutivité pour étendre les logiciels construits avec Mélusine. Seulement, cette évolutivité est définie statiquement avant le démarrage du logiciel. Il serait intéressant de regarder comment faire évoluer un logiciel construit avec Mélusine dynamiquement au cours de son exécution. Cela nécessitera de mettre en place des mécanismes d'insertion dynamique de code ainsi que des interfaces

graphiques cachant la complexité afin de permettre au concepteur du logiciel de construire facilement l'extension qu'il souhaite.

Nous pensons que ce travail représente une contribution importante aux pratiques du génie logiciel tant par ses apports conceptuels tels que nos diverses formes d'extension de domaines conceptuels, que par les apports en terme d'outils, d'environnements et de méthodes, comme l'a montré le développement et l'utilisation de Mélusine en environnement industriel.

Chapitre VII.
Bibliographie[53]

[AAF 02] Arkin A., Askary S., Fordin S., Jekeli W., Kawaguchi K., Orchard D., Pogliani S., Riemer K., Struble S., Takacsi-Nagy P., Trickovic I., Zimek S., "Web Services Choreography Interface (WSCI) 1.0", W3C, 2002.

[ACD 03] Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I., Weerawarana S., "Business Process Specification Language For Web Services", IBM specification, version 1.1, 2003.

[ACID] NuSphere, "ACID Transactions", http://www.nusphere.com/products/library/acid_transactions.htm

[AOSD] Aspect-Oriented Software Development http://aosd.net/

[Ari 00] Ariba, International Business Machines Corporation, Microsoft Corporation, "UDDI Technical White Paper", September 6, 2000

[Ark 02] Arkin A., Intalio, "Business Process Modeling Language", November 13, 2002.

[ASCMK 03] Anyanwu K., Sheth A., Cardoso J., MillerJ., Kochut K., "Healthcare Enterprise Process Development and Integration", Journal of Research and Practice in Information Technology, Vol. 35, No. 2, May 2003. http://www.acs.org.au/jrpit/JRPITVolumes/JRPIT35/JRPIT35.2.83.pdf

[AT 93] Action Workflow System product literature. Action Technologies Inc., 1993.

[Bar 92] Barghouti N. S., "Supporting Cooperation in the Marvel Process-Centered SDE", in: H. Weber (Ed.), Fifth ACM SIGSOFT Symposium on Software Development Environments, Vol. 17 of Special issue of Software Engineering Notes, Tyson's Corner VA, 1992, pp. 21-31.

[53] Certaines références bibliographiques présentées dans ce document sont des adresses URLs (Uniform Resource Locator) de pages web contenant l'information référencée au moment de la rédaction de ce document. Cependant, j'attire l'attention du lecteur au fait que le contenu de ces pages web peut être modifié dans le temps.

[BB 02] Bézivin J., Blanc X., "MDA : vers un important changement de paradigme en génie logiciel". Développeur Référence. Juillet 2002.

[BB 04] Bézivin J., Breton E., "Applying the basic principles of model engineering to the field of process engineering". CEPIS, UPGRADE, The European Journal for the Informatics Professional : Vol. V, No. 5, (2004)

[BBFGL 94] Bandinelli S., Barga M., Fuggetta A., Ghezzi C., Lavazza L., "SPADE An Environment for Software Process Analysis, Design and Enactment". Software Process Modeling and Technology, eds. A. Finkelstein,J. Krammer, B. Nuseibeh, Research Studies Press, Taunton, (1994).

[BE 87] Belkhatir N., Estublier J., "Software management constraints and action triggering in Adele program database". In 1st European Software Engineering Conf., pages 47–57, Strasbourg, France, Sept. 1987.

[BEM 91] Belkhatir N., Estublier J., Melo W. L., "Software process modeling in Adele : The ISPW-7 example". In I. Thomas, editor, Proc. of the 7th Int'l Software Process Work-shop, San Francisco, CA, October 16–18 1991. IEEE Computer Society Press.

[Bez 02] Bézivin J., "From Object Composition to Model Transformation with the MDA". TOOLS USA, (August 2001), Santa Barbara, USA. Available at http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda

[BGR 94] Bruynooghe R.F., Greenwood R.M., Robertson I., Sa J., Snowdon R.A., and Warboys B.C., "PADM : Towards a total process modelling system". In A. Finklestein, J. Kramer, and B. Nuseibeh, editors, Software Process Modelling and Technology, pages 293–334. Research Studies Press, 1994.

[BizTalk] "Microsoft BizTalk Server Product Overview" http://www.microsoft.com/biztalk/evaluation/overview/biztalkserver.mspx

[Biz 04] "Microsoft BizTalk Server 2004", "BizTalk Server 2004 Conceptual Overview", Microsoft, 2004. msdn.microsoft.com/library/en-us/introduction/htm/ebiz_intro_story_irfb.asp

[BK 94] Ben Shaul I. Z., Kaiser G. E., "A paradigm for decentralized process modelling and its realization in the oz environment", In Proceedings of 16th Int'l Conference on Software Engineering, pages 179–190, Sorrento, Italia, May 1994.

[BT 96] Bolcer G. A., Taylor R. N., "Endeavors: A Process System Integration Infrastructure". 4th. International Conference on Software Process ICSP '96, (December 2-6, 1996).

[CGVBFH 02] Chessell M., Griffin C., Vines D., Butler M., Ferreira C., Henderson P., "Extending the concept of transaction compensation", IBM Systems Journal, VOL 41, NO 4, 2002.

[Centr'Actoll] Projet Centr'Actoll
http://www-adele.imag.fr/Les.Groupes/centractoll/

[Ch 02] Chauvet J.-M., "Services Web avec SOAP, WSDL, UDDI, ebXML...", Eyrolles 2002, ISBN : 2212110472

[Choreo] "ChoreoServer" product overview.
http://www.openstorm.com/overview.shtml

[CM ISAM] Carnegie Mellon, Software Engineering Institute (SEI), "TSP and the Integrated Software Acquisition Metrics (ISAM) Project"
http://www.sei.cmu.edu/tsp/isam.html

[CM PSP] Carnegie Mellon, Software Engineering Institute (SEI), "Personal Software Process (PSP)"
http://www.sei.cmu.edu/tsp/psp.html

[CM SPICE] Carnegie Mellon, Software Engineering Institute (SEI),
"Participating in the SPICE Trials"
http://www.sei.cmu.edu/iso-15504/gettinginvolved/trialhost.html
"ISO/IEC 15504 - An Emerging Standard on Software Process Assessment"
http://www.sei.cmu.edu/iso-15504/moreinformation/seimotives.html

[CM TSP] Carnegie Mellon, Software Engineering Institute (SEI), "Team Software Process (TSP)"
http://www.sei.cmu.edu/tsp/tsp.html

[CMMI] Carnegie Mellon, Software Engineering Institute (SEI), "CMMI" http://www.sei.cmu.edu/cmmi/

[CMMI 03] Carnegie Mellon, Software Engineering Institute (SEI), "Upgrading from SW-CMM to CMMI", white paper, March 2003
http://www.sei.cmu.edu/cmmi/adoption/pdf/upgrading.pdf

[CVK 99] Cosquer J.N., Veríssimo P., Krakowiak S., Decloedt L., "Support for Distributed CSCW Applications". LNCS 1752, p. 295 ff, 1999.

[Der 92] Derniame J.-C. , editor. Proceedings of Second European Workshop on Software Process Technology (EWSPT'92), Trondheim, Norway, September 1992, LNCS 635, Springer-Verlag.

[Der 99] Derniame J.-C., Kaba B., Wastell, D., "Software Process : Principles, Methodology and Technology". Springer-Verlag, Lecture Notes in Computer Science 1500, 1999.

[DLR 04] De La Rosa R., "Découverte et Sélection de Services Web pour une application Mélusine", Rapport de Master Recherche 2ème année (DEA) Informatique : Systèmes d'information, Université de Grenoble. Septembre 2004.

[EAD 99] Estublier J., Amiour M., Dami S., “Building a Federation of Process Support System”, Conference on Work Activity Coordination and Cooperation (WACC), ACM SIGSOFT, Vol. 24, No. 2, San Francisco, USA, February 1999.

[ECB 98] Estublier J., Cunin P.Y., Belkhatir N., “Architectures for Process Support System Interoperability”, ICSP 5, pp. 137-147, Chicago, Illinois, USA, June 1998.

[ED 96] Estublier J., Dami S., “Process Engine Interoperability : An experiment”, In C. Montangero, editor, European Workshop on Software Process Technology (EWSPT5), LNCS, Nancy, France, October 9-11 1996. Springer Verlag, pages 43--61.

[EDA 98] Estublier J., Dami S.,Amiour M., “APEL : a Graphical Yet Executable Formalism for Process Modeling”, Automated Software Engineering, ASE journal. Vol. 5, Issue 1, 1998.

[EDMD 97] Emam K. E., Drouin J.-N., Melo W., Dorling A., "SPICE : The Theory and Practice of Software Process Improvement and Capability Determination" , Wiley-IEEE Computer Society Pr, ISBN: 0818677988, October 1997

[EGR 91] Ellis C.A., Gibbs S.J., Rein G.L., "Groupware, Somme Issues and Experiences", Communications of the ACM, 34(1), pp 38-58, 1991.

[EIV 05] Estublier J., Ionita A.I., Vega G., "A Domain Composition Approach", Published in 2005 International Workshop on Applications of UML/MDA to Software Systems(UMSS’05) SERP'05, June 2005, Las Vegas, USA.

[ELC 02] Estublier J., Leblang D., Clemm G., Conradi R., VanDerHoek A., Tichy W., "The Impact of the research community in the field of Software Configuration Management", ICSE, May 2002, Orlando, Florida , USA

[ELCA 05] ELCA-a leading Swiss IT solutions provider, "SPICE : modèle de maturité" http://www.elca.ch/Competencies/Quality/spice.php

[Est 00] Estublier J., "Software Configuration Management : A Roadmap", In Proceedings of 22nd International Conference on Software Engineering, The Future of Software Engineering, pp. 279-289, ACM Press, 2000.

[ES 05a] Estublier J., Sanlaville S., "Business Processes and Workflow Coordination of Web Services", IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05), March 2005, Hong Kong

[ES 05b] Estublier J., Sanlaville S., "Extensible Process Support Environments for Web Services Orchestration", International IEEE Conference on Next Generation Web Services Practices (NWeSP'05), August 2005, Seoul, Korea.

[ES 05c] Estublier J., Sanlaville S., "Extensible Process Support Environments for Web Services Orchestration", International Journal of Web Services Practices (IJWSP), December 2005. Selected article at the NWeSP'05.conférence.

[EVLSV 03] Estublier J., Villalobos J., Le A-T., Sanlaville S., Vega G., "An Approach and Framework for Extensible Process Support System", 9th European Workshop on Software Process Technology (EWSPT 2003), September 2003, Helsinki, Finland. LNCS 2786, ISBN :3-540-40764-2.

[FKN 94] Finkelstein A., Kramer J., Nuseibeh B., editors. Software Process Modelling and Technology. Research Studies Press Ltd., Taunton, Somerset, U.K., 1994.

[Fra 02] Francou L., "APEL EDITOR V 2.0 - Mini tutorial", Rapport Technique Équipe Adèle-laboratoire LSR, décembre 2002.

[Frye 94] Frye C., "Move to Workflow Provokes Business Process Scrutiny", Software Magazine, April, 1994.

[GHS 95] Georgakopoulos D., Hornick M.F., Sheth A.P., “An overview of workflow management-from process modeling to workflow automation infrastructure”. Distributed and Parallel Databases 3(2) : 119-153, 1995.

[GJ 92] Gruhn V., Jegelka R., "An Evaluation of Funsoft Nets", pages 196–214. In Derniame [Dern92], September 1992.

[God 02] Godart C., "Tutorial : Les outils du travail coopératif. Un point de vue ingénierie des données". 18ème Journées Bases de Données Avancées - BDA'02, Evry, France, Octobre 2002.

[Gri ISC] Griffault A, Université Bordeaux 1 - Master S&T Informatique - Spécialité Ingénierie des Systèmes Critiques
http://dept-info.labri.fr/~griffaul/Administration/Master/PagesWebISC/

[HG 03] Hu J., Grefen P., "Conceptual framework and architecture for service mediating workflow management", Information & Software Technology, 45 (13) : 929-939, 2003.

[HTTP] “Hypertext Transfer Protocol - HTTP/1.1”, W3C, 1999
http://www.w3.org/Protocols/rfc2616/rfc2616.html

[IBM 02] IBM alphaWorks, "BPWS4J A platform for creating and executing BPEL4WS processes", 9 Aout 2002 .
http://www.alphaworks.ibm.com/tech/bpws4j

[IEEE-STD] IEEE Std 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology," IEEE Computer Society, September 1990.

[Int n3 prod] Intalio|n³ product
http://www.intalio.com/products/index.xpg

[Int n3 01] "Intalio n3 BPMS", BPMI.org. SAN MATEO, CA - March 27, 2001 http://www.bpmi.org/downloads/2001-03-27-Intalio.pdf

[JPSW 94] Junkermann G., Peuschel B., Schaefer W., Wolf S., "MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment", pages 103–129. In Finkelstein et al. [FKN 94], 1994.

[JS 02] Java Skyline, "Collaxa 2.0 BPEL4WS Server", 11 Mars 2002 http://www.javaskyline.com/20030311_collaxa.html

[Kai 98] Kaiser G., "Experience with Marvel", ISPW 1989: 82-84, Kennebunkport, Maine, USA

[KCH 04] Keen M., Cavell J., Hill S., Kee C.K., Neave W., Rumph B., Tran H., "BPEL4WS Business Processes with WebSphere", "Business Integration : Understanding, Modeling,Migrating", IBM Redbook, December 2004

[Kin 00] Kinnula A., “Software process engineering systems : models and industry cases”, session 3.2. “Quality Improvement Paradigm –cycle”, Department of Information Processing Science, University of Oulu, 29 February 2000.
http://herkules.oulu.fi/isbn9514265084/html/x287.html

[KLB 01] Kuno, H., Lemon, M., Beringer, D., "Using WSCL in a UDDI Registry : UDDI Best Practices Document". May 2001

[KM 03] Kadima H., Monfort V., "Les Web services - Techniques, démarches et outils : XML-WSDL-SOAP-UDDI-Rosetta-UML", Dunod/01 Informatique, 2003.

[KPB 94] Kaiser G. E., Popovich S. S., Ben-Shaul I. Z., "A bi-level language for software process modeling". In W. Tichy editor, Configuration Management, John Wiley and Sons Ltd, 1994.

[Kru 00] Kruchten P., "The rational unified process : an introduction", Collection : Addison-Wesley object technology series, 2000.

[Le 04] Le A.T., "Fédération : une architecture logicielle pour la construction d'applications dirigée par les modèles", Thèse de doctorat, Université Joseph Fourier, janvier 2004.

[Les 03] Lestideau V., "Modèles et environnement pour configurer et déployer des systèmes logiciels", Thèse de doctorat, Université de Savoie, décembre 2003.

[Ley 01] Leymann F., "Web Services Flow Language (WSFL 1.0)", IBM Software Group, May 2001

[Mc 92] McCready S., "There is more than one kind of Workflow Software", Computerworld, November 2, 1992.

[MIME] "Multipurpose Internet Mail Extensions (MIME) Part Two : Media Types", RFC 2046, N. Freed, N. Borenstein, November 1996.
http://www.rfc-editor.org/rfc/rfc2046.txt

[Momentum] La société "Momentum SI"
http://www.momentumsi.com/

[Mom Perf] PerfectXML, "Momentum Software Inc.".
http://www.perfectxml.com/411/DirDetails.asp?ID=193

[MS Project] Microsoft Office Project
http://www.microsoft.com/france/office/project/prodinfo/overview.asp

[Myerson] Myerson J., "Web Service Architectures", Web Service Architect Journal.
http://www.webservicesarchitect.com

[OMG] Object Management Group
http://www.omg.org/

[OP 04] Oracle Press, "Oracle finalise la première plate-forme complète pour SOA et l'intégration et annonce Oracle BPEL Process Manager - Oracle acquiert le premier moteur BPEL natif du marché grâce au rachat de Collaxa", 29 juin 2004.
http://www.oracle.com/global/fr/corporate/press/collaxa.html

[OpenStorm] La société "OpenStorm"
http://www.openstorm.com/overview.shtml

[PCCW 93] Paulk M., Curtis B., Chrissis M., Weber C., "Capability Maturity Model for Software", Version 1.1, Technical Report CMU/SEI-93-TR-24, 1993

[PCCW 93 fr] Paulk M., Curtis B., Chrissis M., Weber C., "Modèle d'évolution des capacités logiciel", Version 1.1, CMU/SEI-93-TR-24, ESC-TR-93-177, Fevrier 1993. Traduction réalisée par le Centre de génie logiciel appliqué - une division du Centre de recherche informatique de Montréal - CRIM

[Pel 03] Peltz C., "web services orchestration a review of emerging technologies, tools, and standards", Hewlett Packard, Co. January 2003.

[PK 98] Port D., Kaiser G., "Collaborative Technologies for Evolving Software Systems", IEEE Internet Computing, vol. 02, no. 6, pp. 79-83, November/December 1998.

[PMI 96] Project Management Institute, "A Guide to the Project Management Body of knowledge", 1996

[RBPEL 91] Rumbaugh J., Blaha M., Premerali W., Eddy F., Lorensen W., "Object-Oriented Modelling and Design", Prentice-Hall, Emglewood Cliffs, New Jersy, 1991.

[Rid 91] Riddle W.E., "Activity Structure Definitions". Technical Report 7-52-3, Software Design & Analysis, March, 1991.

[RMB 01] Ruh W.A., Maginnis F.X., Brown W.J., "Enterprise Application Integration : A Wiley Tech Brief". John Wiley and Sons, Inc, 2001.

[Roq 02] Roques P., "UML : modéliser un site e-commerce", Collection : Les cahiers du programmeur, Paris : Eyrolles, 2002.

[Sat 01] Satish T., “XLANG : Web Services for Business Process Design”, Microsoft, 2001.
http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[SE 05] Sanlaville S., Estublier J., "Melusine : un environnement de modélisation et de coordination", Revue d'Ingénierie des Systèmes d'Information, numéro spécial "Services Web, théories et applications",2005, Volume 10, Numéro 3, RSTI-ISI-10/2005 (pages 29-48), ISBN :2-7462-1195-5

[SEL GQM] Software Engineering Laboratory (SEL), "Goal-Question-Metric (GQM)". http://sel.gsfc.nasa.gov/website/exp-factory/gqm.htm

[Shaw 90] Shaw M., "Prospects for an Engineering Discipline of Software", IEEE Software, November 1990, pp. 15-24

[SOAP] "Simple Object Access Protocol (SOAP) 1.1", W3C Note, 24 June 2003 http://www.w3.org/TR/soap/

[SOE] Service Oriented Enterprise,
http://www.serviceoriented.org/web_service_orchestration.html

[Som 04] Sommerville I., "Software engineering", 7th edition. Addison-Wesley, ISBN 00321210263, 2004

[SPEM 05] Software Process Engineering Metamodel Specification - Version 1.1, OMG, formal/05-01-06 , January 2005

[Spi 05] Spicer J., "Oracle Jumps to the Forefront of BPEL", Oracle Magasine, May/June 2005.
http://www.oracle.com/technology/oramag/oracle/05-may/o35editor.html

[UDDI] Industry Initiative "Universal Description, Discovery and Integration" http://www.uddi.org/specification.html

[VdA patt] Van der Aalst W.M.P., "workflow patterns page"
http://is.tm.tue.nl/research/patterns/

[VDT 02] Van der Aalst W.M.P., Dumas M., Ter Hofstede A.H.M., Wohed P., "Pattern Based Analysis of BPML (and WSCI)", QUT Technical Report,FIT-TR-2002-05. Queensland University of Technology, Brisbane, 2002.

[Vil 02] J. Villalobos. "APEL : Spécification formelle du moteur". Rapport Technique Équipe Adèle. mars 2002

[Vil 03] Villalobos J., "Fédération de Composants : une Architecture Logicielle pour la Composition par Coordination", Thèse de doctorat, Université Joseph Fourier, juillet 2003.

[VT 02] Van der Aalst W.M.P., Ter Hofstede A.H.M., "YAWL : Yet Another Workflow Language", QUT Technical report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.

[VTK 02] Van der Aalst W.M.P., Ter Hofstede A.H.M., Kiepuszewski B., Barros A.P., "Workflow Patterns", QUT Technical report. FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002. (To appear in Distributed and Parallel Databases)

[VT YAWL] Van der Aalst W.M.P., Ter Hofstede A.H.M., “YAWL home page” http://www.yawl.fit.qut.edu.au/

[WfMC] Workflow Management Coalition
http://www.wfmc.org/

[Wiki GP] http://fr.wikipedia.org/wiki/Gestion_de_projet

[Wil 01] Wile D. S., "Supporting the DSL Spectrum", Journal of Computing and Information Technology, CIT 9, 2001 (4) 263-287.

[WSCL 02] Web Services Conversation Language (WSCL) 1.0, W3C Note 14 March 2002
http://www.w3.org/TR/2002/NOTE-wscl10-20020314/

[WSDL 01] Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001
http://www.w3.org/TR/wsdl

[WS MQ] "WebSphere MQ", IBM
http://www-306.ibm.com/software/integration/wmq/

[WVD 02] Wohed P., Van der Aalst W.M.P., Dumas M., Ter Hofstede A.H.M., "Pattern-Based Analysis of BPEL4WS". , QUT Technical report, FIT-TR-2002-04. Queensland University of Technology, Brisbane, 2002.

[XML] Extensible Markup Language, "The XML Specification", W3C Recommendation, 04 February 2004
http://www.w3.org/TR/REC-xml/

[XSD] XML Schema, W3C Recommendation, 28 October 2004
http://www.w3.org/TR/xmlschema-0/