INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

|__|__|__|__|__|__|__|__|__|__|

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT POLYTECHNIQUE DE GRENOBLE

Spécialité : INFORMATIQUE

préparée au LABORATOIRE D'INFORMATIQUE DE GRENOBLE

dans le cadre de l'**École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

présentée et soutenue publiquement

par

GUILLAUME GODET-BAR

le 30 novembre 2009

SPÉCIFICATION ET OUTILLAGE D'UNE MÉTHODE DE CONCEPTION DES SYSTÈMES DE RÉALITÉ MIXTE

Sous la direction de DOMINIQUE RIEU

et SOPHIE DUPUY-CHESSA

JURY

PR. JACQUES MOSSIÈRE ..... Président
PR. ÉRIC DUBOIS ..... Rapporteur
MR. PHILIP GRAY ..... Rapporteur
PR. DOMINIQUE RIEU ..... Directeur de thèse
DR. SOPHIE DUPUY-CHESSA ..... Co-directeur de thèse
PR. CAMILLE ROSENTHAL-SABROUX ..... Examinateur
DR. CHRISTIAN BASTIEN ..... Examinateur

Institut polytechnique de Grenoble

Thèse

pour obtenir le grade de

Docteur de l'Institut polytechnique de Grenoble

Spécialité : Informatique

préparée au Laboratoire d'Informatique de Grenoble

dans le cadre de l'**École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

présentée et soutenue publiquement

par

Guillaume Godet-Bar

le 30 novembre 2009

Spécification et outillage d'une méthode de conception des systèmes de réalité mixte

Sous la direction de Dominique Rieu

et Sophie Dupuy-Chessa

Jury

Pr. Jacques Mossière ..... Président
Pr. Éric Dubois ..... Rapporteur
Mr. Philip Gray ..... Rapporteur
Pr. Dominique Rieu ..... Directeur de thèse
Dr. Sophie Dupuy-Chessa ..... Co-directeur de thèse
Pr. Camille Rosenthal-Sabroux ..... Examinateur
Dr. Christian Bastien ..... Examinateur

Table des matières

Table des figures

Liste des tableaux

Introduction

Mark Weiser [123, 124] consacre, au début des années 90, la vision d'une informatique intuitive, ambiante et pervasive, aux antipodes des stations de travail encombrantes utilisées alors. Concrétisation de cette vision, les interfaces de réalité mixte – mêlant éléments virtuels et objets physiques –, autrefois cantonnées aux laboratoires, émergent comme objets du quotidien.

Des études académiques toujours plus nombreuses confirment l'efficience des interfaces de réalité mixte pour des tâches dédiées : les tâches de recherche topologique utilisant la manipulation gestuelle d'une carte affichée sur une table tactile sont systématiquement plus efficace que les mêmes tâches, exécutées avec des interfaces classiques basées sur l'écran, le clavier, et la souris. Ces dernières années, la diffusion toujours plus large d'applications utilisant la réalité mixte transforme ce secteur, auparavant ultra-spécialisé et fortement lié au monde académique, en une industrie de la consommation courante.

Ce phénomène procède, selon nous, de deux dynamiques : d'une part, l'industrie du divertissement commercialise des produits dédiés à des interactions augmentées ; d'autre part, des produits initialement non destinés aux interactions de réalité mixte, mais intégrant plusieurs technologies de capteurs programmables (accéléromètres, gyroscopes, puces GPS…) catalysent la création de projets d'interaction de réalité mixte.

Il existe donc un ensemble de pratiques et savoir-faire, issus des laboratoires, en train de s'organiser et de se structurer. Par ailleurs, la démocratisation de capteurs fiables permettant de capter la localisation et les gestes de l'utilisateur, intégrés dans des dispositifs domestiques ou mobiles, rend accessible le développement d'application de réalité mixte au plus grand nombre. Ajoutons que des infrastructures matérielles et logicielles pervasives (accès Internet haut débit sur les dispositifs mobiles, technologie GPS) complètent l'arsenal de systèmes et interfaces possibles.

Dès lors, considérant les nombreuses qualités des systèmes de réalité mixte et opportunités de développement qu'ils représentent, comment expliquer leur actuelle désaffection par les entreprises ?

Il est indubitable que le développement de systèmes interactifs plus complexes que les classiques formulaires induit plusieurs risques. Les différents dispositifs pouvant être impliqués dans l'interaction mixte et le temps de développement de telles solutions constituent tout d'abord un risque financier important. De plus, les contraintes ergonomiques de ces systèmes sont nombreuses et propres à ces derniers, dont la négligence peut entraîner des inconforts fatals à leur adoption. Enfin, les systèmes de réalité mixte représentent une technologie dont la nouveauté peut dérouter certains utilisateurs. La maîtrise de ces risques est

donc indispensable pour la pérennité de ces systèmes dans les entreprises.

Des domaines de conception antinomiques ?

Les applications d'entreprise ne sont que rarement utilisées sans intégration dans le système d'information global. Elles sont le plus souvent centrées sur les fonctionnalités, et constituent l'extension opérationnelle d'un système d'information construit sur trois niveaux de mise en œuvre : l'intention, l'organisation et l'opérationnalisation. Les méthodes de développement utilisées pour la construction de ces applications reflètent plus ou moins fidèlement ce découpage. Inversement, ces méthodes ne traitent l'utilisabilité des systèmes développés que marginalement. Il n'est donc malheureusement pas rare de voir des applications répondant aux besoins fonctionnels de ses utilisateurs, mais dont les fonctionnalités sont obscurcies, ou même cachées, par une interface à l'ergonomie plus que perfectible.

Par ailleurs, les rares méthodes de développement des interfaces classiques, et celles destinées au développement des interface de réalité mixte, plus rares encore, considèrent exhaustivement l'utilisabilité des solutions et en délaissent les aspects fonctionnels. Ceux-ci sont laissées à la charge hypothétique des spécialistes du génie logiciel, sans pour autant guider l'intégration des modèles et solutions IHM produits.

Malgré leurs divergences, ces deux approches peuvent-elles être conciliées, et ce sans compromettre la qualité des pratiques de conception élaborées au sein de l'un et l'autre domaine ? Sur quelle base ancrer des collaborations entre des acteurs du développement issus de cultures informatiques, et donc de pratiques, différentes ? Comment capitaliser sur ces processus, modèles, langages et outils élaborés dans des contextes hermétiques l'un à l'autre ?

Objectifs et démarche

L'objectif général de notre thèse est de proposer une méthode de développement pour les systèmes de réalité mixte et les interfaces classiques :
- répondant aux besoins fonctionnels des utilisateurs ;
- utilisables d'un point de vue ergonomique ;
- intégrables dans des écosystèmes d'entreprise ;
- reposant sur des choix de conception traçables.

Bien que couvrant un spectre large, nous estimons qu'il est nécessaire d'embrasser les différents aspects de cette problématique hybride pour qu'une contribution fasse sens.

Dans un premier temps, nous analyserons les spécificités des systèmes de réalité mixte, dont nous tenterons de déduire des besoins méthodologiques propres à leur conception. Nous évaluerons l'adéquation entre les pratiques d'intégration du génie logiciel et de l'ingénierie de l'interaction homme-machine existantes, et les besoins méthodologiques identifiés.

Si cette étude laisse apparaître des carences en termes des processus, modèles, langages ou outils de conception de la littérature, nous proposerons des solutions conceptuelles à ces différents niveaux. Il s'agira par la suite de mettre en œuvre ces propositions, sous la forme d'une méthode de conception originale. Considérant l'ampleur de la tâche, nous recourrons à une méthode existante et éprouvée, dans laquelle nous intégrerons la traduction pratique

de nos principes.

Nous traiterons enfin la validation de nos contributions. Dans le contexte des méthodologies de conception, l'évaluation concerne à la fois le processus, les modèles, les langages et les outils employés, auquel s'ajoute l'acceptation par les utilisateurs.

Contributions

Nos travaux de thèse introduisent une méthode complète et non prescriptive pour la conception de systèmes exploitant une interface classique ou de réalité mixte, intégrés dans un système d'information pré-existant. Cette méthode permet d'apporter une valeur ajoutée à la fois aux fonctionnalités développées, grâce à une analyse métier, conceptuelle et organisationnelle éprouvée, ainsi qu'à l'utilisabilité des solutions, en intégrant les pratiques et modèles de l'interaction homme-machine.

En complément de notre méthode, nous avons élaboré plusieurs outils facilitant son opérationnalisation : AGAP Lite, un outil pour la documentation du processus ; Sonata, un intergiciel simplifiant l'implémentation et l'exécution des produits de la méthode ; un ensemble de transformations automatisant les activités systématiques de la méthode.

Organisation du mémoire

Ce mémoire reprend les trois points de notre démarche de recherche.

Nous décrivons en **première partie** les éléments conceptuels sur lesquels s'appuient nos travaux :

- le **chapitre 1** présente le domaine des systèmes de réalité mixte, ainsi que les processus et modèles du domaine de l'interaction homme-machine employés pour la conception des interfaces classiques et des systèmes de réalité mixte ;
- le **chapitre 2** décrit l'analyse qualitative des différentes méthodes mêlant pratiques du génie logiciel et de l'interaction homme-machine ;
- le **chapitre 3** introduit Symphony, une méthode de conception issue de la communauté du génie logiciel, et sur laquelle nous basons nos contributions.

Dans la **deuxième partie** de ce mémoire, nous abordons les contributions théoriques apportées au domaine de la conception des interfaces de réaltié mixte :

- le **chapitre 4** détaille les principes sur lesquels nous construisons une méthodologie de conception pour les systèmes de réalité mixte ;
- le **chapitre 5** applique ces principes sur un cas d'étude : la gestion des états des lieux par une agence immobilière.

La **troisième partie** est consacrée aux réalisations logicielles et aux évaluations :

- le **chapitre 6** introduit Sonata : un intergiciel facilitant l'exécution des systèmes développés suivant la méthode Symphony étendue ;
- le **chapitre 7** développe les modèles et transformations de modèles employés pour l'automatisation des aspects systématiques de la conception ;

- le **chapitre 8**, enfin, expose les évaluations qualitatives pratiquées sur nos contributions théoriques et logicielles.

En conclusion, nous synthétisons nos contributions, identifions les limites de nos travaux et proposons de nombreuses perspectives d'ouverture.

Trois annexes complètent ce mémoire :

- l'**annexe A** décrit la structure de l'intergiciel Sonata ;
- l'**annexe B** présente l'implémentation des règles de transformation pour l'automatisation de certaines activités de conception ;
- l'**annexe C** recense les résultats de l'une des évaluations de la méthode.

Première partie

Etat de l'art et outils conceptuels pour les méthodes de développement

CHAPITRE 1

Systèmes de réalité mixte : concepts, modèles et processus

> The future is here. It's just not widely distributed yet.
>
> — William Gibson

Parce que nous nous attaquons à la problématique de la conception des systèmes de réalité mixte (SRM), il convient de préciser quelques principes fondamentaux du développement des interfaces homme-machine, et plus particulièrement des modèles et processus qui le constituent. Subséquemment, cette structure conceptuelle nous permet alors d'aborder le domaine plus spécifique qu'est la réalité mixte.

1.1 Champ d'étude des systèmes de réalité mixte

Après avoir présenté deux exemples représentatifs de systèmes exploitant les techniques d'interaction en réalité mixte, nous définissons les principes essentiels de ce domaine, puis en identifions les spécificités. Par la suite, nous décrivons plusieurs modèles et processus utilisés dans le cadre de la conception des interfaces homme-machine classiques et de réalité mixte.

1.1.1 Exemples de systèmes de réalité mixte

Afin d'illustrer quelques types d'interaction en réalité mixte existants, nous nous appuyons sur deux applications existantes, présentées ci-dessous.

La figure 1.1 présente un exemple d'application commerciale de la réalité mixte : le jeu *Eye of Judgement*©. Le principe de ce système est simple : deux joueurs s'affrontent sur une surface de jeu découpée en neuf zones, sous l'œil d'une caméra permettant au système de retranscrire

(a) Dispositifs physiques

(b) Vue composite affichée sur l'écran

FIGURE 1.1 – Virtualité augmentée sur un téléviseur : le jeu *Eye of Judgement*©

le déroulement de la partie sur un téléviseur. L'appropriation de chaque zone fait l'objet d'un affrontement, matérialisé par des cartes à jouer placées sur la surface de jeu par chaque joueur. Les cartes sont toutes marquées d'un code-barre bidimensionnel identifiant celles-ci de manière unique. La caméra permet au système de déterminer la position et l'orientation des cartes, grâce à la déformation spaciale de leur code-barre, subséquente à l'orientation et la position de la carte par rapport à la caméra. Outre sa fonction de localisation, chaque code-barre identifie une créature fantastique, que le système représente en trois dimensions sur l'écran de télévision, comme posée sur sa carte à jouer (figure 1.1(b)).

La figure 1.2 présente une nouvelle application commerciale de la réalité mixte, cette fois-ci en situation de mobilité : *Nearest Tube*©. Cette application propose à un utilisateur se déplaçant dans la ville de Londres les stations de métro les plus proches. Afin de faciliter la tâche de l'utilisateur, le système utilise à la fois les données géomagnétiques et GPS ainsi que l'orientation de l'appareil, toutes deux fournies par des sous-systèmes matériels du dispositif. L'application utilise le capteur vidéo situé au dos du dispositif mobile pour afficher la partie de l'environnement physique de l'utilisateur sur laquelle celui-ci oriente l'appareil et y superposer des données numériques. Lorsque l'utilisateur tient l'appareil parallèle au sol, une carte du métro londonien apparaît, orientée en fonction de la position du dispositif. Lorsque l'utilisateur oriente l'appareil vers les rues qui l'entourent, des marqueurs indiquant

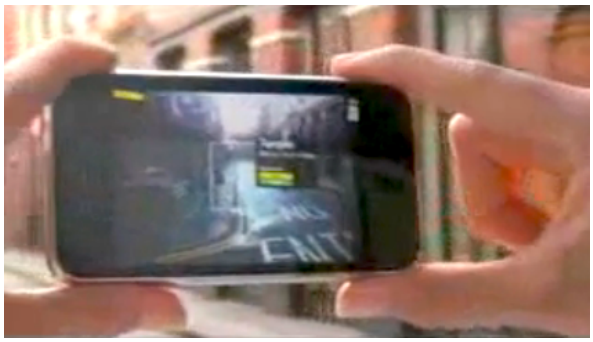


Figure 1.2 – Réalité augmentée sur un dispositif mobile : l'application *Nearest Tube*©

les stations de métro les plus proches apparaissent (situation représentée sur la figure 1.2) dans la direction desdites stations.

Les données relatives aux stations de métro sont obtenues, pour une partie, grâce à la connexion du dispositif à des services web. L'instauration de la réalité mixte sur dispositif mobile s'appuie ainsi sur l'accès pervasif à l'information, mis en œuvre par une infrastructure de services matériels (serveurs, connexions mobiles à haut débit…) et logiciels (protocoles de communication, algorithmes de localisation, systèmes d'information…) très large.

Les deux exemples décrits ci-dessus exploitent un nombre important de concepts de la réalité mixte, parmi lesquels les notions de dispositif, de modalité, de technique d'interaction, de réalité et de virtualité augmentée, que nous nous employons à définir plus précisément ci-dessous.

1.1.2 Définitions

Considérons d'abord le terme de réalité mixte lui-même, lequel désigne l'ensemble des applications dans lesquelles des éléments multimédias virtuels (séquences animées, images, sons, signaux haptiques etc.) se superposent harmonieusement au monde physique. Dans ce contexte d'interaction encore inhabituel, les modes classiques d'interaction avec le système – par exemple des commandes préétablies tapées sur un clavier, ou bien le déplacement d'un pointeur sur un écran – sont inadaptés, et doivent être remplacés par d'autres types d'interaction.

Une pléïade de nouveaux dispositifs investissent le champ de l'interaction personne-système : les exemples ci-dessus ont recours à des accéléromètres captant les gestes de l'utilisateur, des caméras permettant le suivi des gestes et l'identification d'objets mixtes, des surfaces tactiles, ainsi qu'à des capteurs GPS.

En entrée comme en sortie de ces nouveaux dispositifs, de nouvelles modalités de communication entre dispositifs, objets physiques et utilisateur émergent. Le terme de modalité est compris ici au sens donné par Coutaz et Nigay [81] comme un couple $< d, l >$ tel que :

- d désigne un dispositif physique, tel que la souris ;
- l désigne un langage d'interaction, constitué d'éléments terminaux et d'une grammaire, par exemple les déplacements relatifs de la souris ou bien le langage $< clicDroit, clicGauche >$.

Par la suite, on distinguera les modalités dites actives des modalités dites passives [91]. Les premières dénotent les actions explicites de l'utilisateur sur le système et les réactions

émises par le système. Les secondes désignent les données implicites captées par le système, telles que la position de l'utilisateur, l'orientation du nord magnétique etc.

Ces différentes modalités instaurent des interactions avec de nouveaux objets, qui cumulent une réalité physique – par exemple, la carte à jouer sur laquelle est imprimé un code-barre –, avec une responsabilité au sein d'un système numérique, – suivant notre exemple, la localisation d'une créature fantastique numérique.

Modalités et objets sont ensuite intégrés en techniques d'interaction, soit *« une manière d'utiliser des dispositifs physiques d'entrée/sortie afin de réaliser une tâche générique dans le contexte d'un dialogue homme-machine »* [1] [50]. A. B. Tucker offre une variante de cette définition, précisant qu'une technique d'interaction est une *« fusion d'entrées et de sorties, consistant de tous les éléments logiciels et matériels permettant à l'utilisateur d'accomplir une tâche »* [2] [120].

Ces deux définitions laissent une certaine marge d'interprétation quant à la granularité du concept de « tâche générique ». L'interaction décrite dans *Nearest Tube*© peut ainsi être considérée comme une seule technique, remplissant la tâche générique « exploiter des données numériques superposées aux données réelles », ou bien de deux tâches « superposer une carte numérique sur un lieu réel » et « positionner des marqueurs numériques sur des lieux réels », selon la problématique plus générale motivant l'analyse des techniques d'interaction.

La réalité mixte constitue donc un paradigme d'interaction à part entière, composé de dispositifs multiples et hétérogènes, de techniques d'interaction spécifiques et d'artefacts hybrides, entre objets physiques et entités virtuelles. Elle a notamment pour fonction d'étendre les capacités de l'utilisateur, tout en effaçant progressivement de son champ de perception les infrastructures informatiques sous-jacentes, selon la vision de Weiser [123].

La réalité mixte recouvre deux approches antinomiques de l'interaction : la réalité augmentée et la virtualité augmentée. E. Dubois [45] les distingue comme suit :

- la réalité augmentée désigne un contexte d'interaction où l'objet de la tâche utilisateur réside dans *le monde physique*, mais où sa réalisation est facilitée par *un système informatique*. L'application *Nearest Tube*© en constituant un exemple type ;
- à l'inverse, la virtualité augmentée désigne un contexte d'interaction où l'objet de la tâche utilisateur est *numérique*, mais où sa réalisation est facilitée par des éléments du *monde physique*. L'application *Eye of Judgement*© en démontre une mise en œuvre.

La figure 1.3 matérialise cette distinction comme deux continua, l'un décrivant un processus de virtualisation du réel (cas de la réalité augmentée), l'autre un processus de réification du virtuel (cas de la virtualité augmentée).

Nous identifions dans la sous-section suivante les éléments de définition spécifiques aux systèmes de réalité mixte.

1. *"An interaction technique is a way of using a physical input/output device to perform a generic task in a human-computer dialogue"*
2. *"An interaction technique is the fusion of input and output, consisting of all software and hardware elements, that provides a way for the user to accomplish a task."*

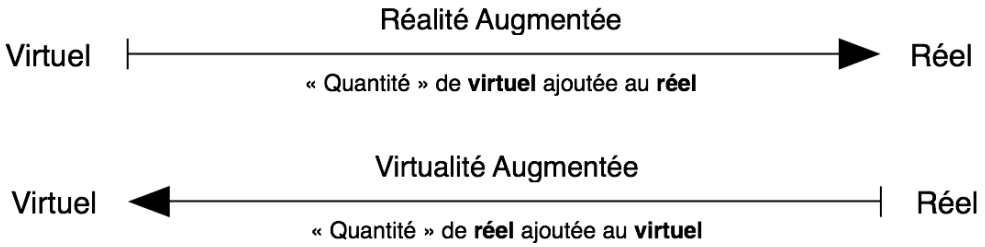


Figure 1.3 – Systèmes de réalité mixte : deux continua distinguant la réalité augmentée de la virtualité augmentée, d'après [45]

1.1.3 Spécificités des systèmes de réalité mixte

Nous retenons dans cet intitulé quelques-uns des points qui distinguent radicalement les applications en réalité mixte des systèmes interactifs traditionnels :

1. Les systèmes de réalité mixte utilisent de nombreux dispositifs hétérogènes, ou tout du moins des sous-systèmes matériels distincts du triptyque clavier–écran–souris. Ces dispositifs capturent aussi bien des données issues de modalités actives que passives ;
2. Chaque application utilise un ensemble spécifique de dispositifs, en fonction de ses fonctionnalités et de son contexte d'utilisation. Il n'existe donc pas d'infrastructure matérielle et/ou logicielle standardisée, pour les systèmes mixtes, bien que des bibliothèques techniques existent (par exemple, ARToolkit [Art]) traitant certains problèmes particuliers (le positionnement d'éléments numériques sur des surfaces marquées d'un motif bidimensionnel, à l'aide d'une caméra) ;
3. Bien que basés sur des dispositifs hétérogènes, certaines fonctionnalités spécifiques au domaine des systèmes mixtes sont récurrentes, tel que la localisation de l'utilisateur ou l'identification d'objets physiques ayant un rôle dans l'interaction. À titre indicatif, une recherche Google sur les termes *« mixed reality user localization »* renvoie à ce jour près de 400 000 résultats. Toutefois, l'implémentation de ces fonctionnalités (et les dispositifs impliqués dans leur réalisation) varie en fonction des spécificités de l'application ;
4. Les flux de données entre les dispositifs d'entrée et de sortie, et le système et l'utilisateur sont une problématique majeure de la conception des systèmes de réalité mixte, contrairement aux systèmes interactifs classiques où les entrées sont assurées par le clavier et la souris, les sorties par l'écran ;
5. Les flux de données en entrée sont plus complexes, car intégrés suivant plusieurs modalités – notamment des capteurs, dont la précision peut être variable ;
6. La superposition d'éléments physiques et numériques correspond à une fusion de modalités en sortie du monde réel et de modalités générées par le système, respectivement. Différents types de composition de modalités (temporel, spatial, articulatoire, syntaxique et sémantique) ont été définis par F. Vernier [122], chacun soumis à des contraintes de continuité d'interaction. La continuité d'interaction est définie au niveau perceptif comme *« la capacité de l'utilisateur à percevoir toutes les données fournies en considérant notamment leur dispersion géographique dans l'environnement ainsi que les sens perceptifs requis par chacune des données »* [45]. Au niveau cognitif, la continuité d'interaction caractérise *« la capacité du système à fournir des données de*

manière à favoriser un interprétation correcte par l'utilisateur » [45]. Dans le contexte de l'application *Nearest Tube*©, la contrainte de continuité d'interaction spatiale consisterait ainsi à s'assurer que l'affichage de la carte du métro londonien est orientée selon le nord magnétique. En ce qui concerne l'application *Eye of Judgement*©, la contrainte de continuité temporelle impose par exemple une forte réactivité du système aux déplacements des cartes à jouer, afin de garantir l'illusion de colocalisation de la carte et de la créature fantastique à laquelle elle correspond.

Considérant ces particularités, les besoins méthodologiques de conception de systèmes de réalité mixte peuvent être ramenés aux points suivants :

1. La complexité des techniques d'interaction utilisées dans les systèmes de réalité mixte, ainsi que les problématiques d'utilisabilité qu'elles soulèvent, requièrent des expertises en conception d'interfaces homme-machines et en ergonomie ;
2. Les pratiques des spécialistes de l'interaction homme-machine se traduisant par la description de modèles de l'interaction (classique et en réalité mixte), il est nécessaire de prendre en compte ces derniers dans la conception ;
3. La conception technique du système tient une place majeure dans le développement des systèmes de réalité mixte, interdépendante de la conception des techniques d'interaction.

La conception des systèmes de réalité mixte s'appuie à la fois sur des modèles et processus génériques du domaine de l'IHM et sur des constructions plus spécifiques. La prise en compte de ces dernières constitue l'objectif de nos travaux.

Nous abordons dans les sections suivantes les notations utilisées pour la conception des interfaces classiques et en réalité mixte, ainsi que les méthodes de conception du domaine de l'interaction homme-machine.

1.2 Modèles pour l'IHM et la réalité mixte

La conception des systèmes de réalité mixte (SRM) intègre à la fois des modèles issus de la conception des interfaces classiques et des modèles traitant de leurs spécificités. Les sous-sections suivantes présentent deux modèles classiques : les scenarii et les modèles de tâches utilisateurs, ainsi qu'une modélisation spécifique.

1.2.1 Modélisation par les scenarii

La notion de scénario, telle que proposée par J. M. Carroll [18, 19] est utilisée depuis quelques décennies pour la conception des interfaces homme-machine. Cette approche contextualisée de l'analyse des besoins est proche de la notion de cas d'utilisation telle que décrite par I. Jacobson [61]. Ce dernier considère d'ailleurs les scenarii comme des instances de cas d'utilisation. Les différences entre ces deux concepts se manifestent d'ailleurs davantage sur la forme que sur le fond, les cas d'utilisation étant généralement plus formels que les scenarii. De plus, contrairement aux cas d'utilisation qui décrivent des utilisations nominales et des cas alternatifs ou d'erreur en annexe du cas principal, les scenarii intègrent indifféremment

cas nominaux, cas alternatifs et cas d'erreur. Par souci de concision, nous nous concentrons ici sur les premiers.

Un scénario est une description d'une facette du système intégrant quatre composants fondamentaux :

- un **contexte** dans lequel se déroule le scénario, qui définit les circonstances réalistes de son application et structure le processus général de création de valeur fourni par le futur système ;
- un ou plusieurs **agents ou acteurs** impliqués dans la réalisation du scénario. Il s'agit ici d'individus, non pas de systèmes informatiques ;
- des **buts ou objectifs** propres à chaque acteur. Un scénario doit au minimum associer un acteur à un objectif. De plus, les objectifs peuvent évoluer au fil du scénario, en réaction à des actions ou événements ;
- une **séquence d'actions ou d'événements** définissant le comportement des acteurs ou l'évolution de l'environnement. Carroll indique que les actions ne sont pas systématiquement reliées à l'objectif de l'acteur les réalisant. Au contraire, elles peuvent dénoter des interruptions au cours de la réalisation de la tâche, les hésitations, le changement d'objectif etc.

J. M. Carroll distingue différents types de scenarii, fonction de leur rôle dans le processus de conception :

- les **scenarii problème**, qui décrivent l'activité actuelle des utilisateurs afin d'identifier les points possibles d'amélioration ;
- les **scenarii d'activité**, aussi appelés **scenarii abstraits**, qui s'appuient sur les tâches utilisateur afin de décrire le niveau intentionnel du futur système, c'est-à-dire, indépendamment des technologies ;
- les **scenarii information et interaction**, aussi appelés **scenarii concrets**, qui décrivent le système sous sa forme finale, concrète.

Nous présentons un exemple de scénario concret dans le tableau 1.1. Les différents composants y sont aisément identifiables : le contexte correspond aux détails du rendez-vous entre Sean et Mary. L'agent du scénario est Sean, son but est de prendre le métro à Victoria Station et la séquence d'actions consiste en trois étapes : lancer l'application, orienter le dispositif autour de l'utilisateur afin d'identifier la station, s'y rendre une fois celle-ci trouvée.

Tableau 1.1 – Exemple de scénario concret pour l'application *Nearest Tube*©

Sean, habitant Oxford, visite son amie Mary à Londres. Il doit la retrouver près de chez elle, dans le quartier de West Kensington. Alors qu'il sort du train à Victoria Station, il sait qu'il doit rejoindre la station de métro également appelée Victoria Station et emprunter la District Line. Devant le fronton de la gare, il lance l'application de repérage, téléchargée sur son dispositif mobile avant son départ. En orientant le dispositif autour de lui, Sean repère rapidement la station grâce au marqueur de position s'affichant sur l'écran. Celle-ci est située à quelques dizaines de mètres de là, sur Buckingham Palace Road, et Sean s'y précipite.

L'accumulation des scenarii permet ainsi de construire les spécifications du système, orientées sur les activités de l'utilisateur.

Dans la sous-section suivante, nous abordons un autre type de représentation des activités

de l'utilisateur, selon une perspective plus radicalement axée sur le domaine de l'interaction homme-machine.

1.2.2 Modélisation des tâches utilisateur

Différentes notations ont été proposées pour représenter les actions de l'utilisateur sur un système interactif. Les premiers modèles de l'interaction homme-machine, tels que GOMS [17] ou UAN [56] se concentrent sur les actions physiques élémentaires de l'utilisateur sur le système, dans une logique prédictive.

Une seconde génération de formalismes ont abordé l'IHM à un plus haut niveau d'abstraction, en considérant les intentions de l'utilisateur comme prémisses aux actions physiques sur le système. On compte parmi ces modèles de tâches utilisateur MAD [101], CTT [93], ou plus récemment K-MAD [24]. Ces modèles sont relativement proches à la fois dans leur expressivité et dans les représentations qu'ils proposent :

- ces modèles proposent une décomposition hiérarchique des tâches, de la tâche la plus abstraite (tâche racine) aux niveaux les plus concrets. Les représentations pyramidales issues de ce type de modélisation ont donné lieu à l'appellation commune d'« arbre de tâches » ;
- aucune limite n'est imposée au concepteur quant au niveau de raffinement maximal des modèles : il est ainsi possible de ne se limiter qu'aux tâches abstraites (intentionnelles) ou bien de préciser jusqu'aux actions élémentaires de l'utilisateur ;
- en termes de tâches physiques, les modèles permettent de décrire, outre les tâches physiques de l'utilisateur (par exemple, se déplacer jusqu'à un automate distributeur de billets), les tâches de communication avec le système (par exemple, rentrer un code d'identification) et les réactions du système (par exemple, fournir à l'utilisateur le solde de son compte en banque) ;
- à un même niveau d'abstraction, les tâches sont liées entre elles par des opérateurs temporels – dénotant le parallélisme, le séquencement ou le recouvrement – et des opérateurs conditionnels – tâches déclenchées à la suite d'une autre, tâches en alternative etc. ;
- chaque modèle propose différents types de décorations permettant d'enrichir la sémantique des arbres de tâches.

Dans le cadre de cette étude, nous adoptons le modèle CTT comme référence pour la modélisation des tâches utilisateur. Ce modèle peut être considéré à ce jour comme le plus répandu. Les paragraphes suivants adoptent donc ce formalisme pour la description des exemples.

Nous retrouvons en figure 1.4 une illustration du formalisme CTT, appliqué à l'application mobile *Nearest Tube*©. Certaines des particularités de ce langage sont représentées : par exemple, les concepts pertinents par rapport à l'interaction sont indiqués en annotation et en italiques. Les opérateurs indiquent, pour le premier niveau de l'arbre de tâches, que les deux tâches abstraites « consulter la carte du métro londonien » et « chercher la station à proximité » sont entrelacées, c'est-à-dire qu'il est possible d'alterner de l'une à l'autre indifféremment, à tout moment. Les opérateurs entre tâches de communication et tâches systèmes, au deuxième niveau de l'arbre de tâches, dénotent l'activation de la tâche système par la tâche de communication : par exemple, il est nécessaire de d'abord orienter le dispositif mobile vers le bas pour obtenir la représentation d'une carte orientée du métro londonien.

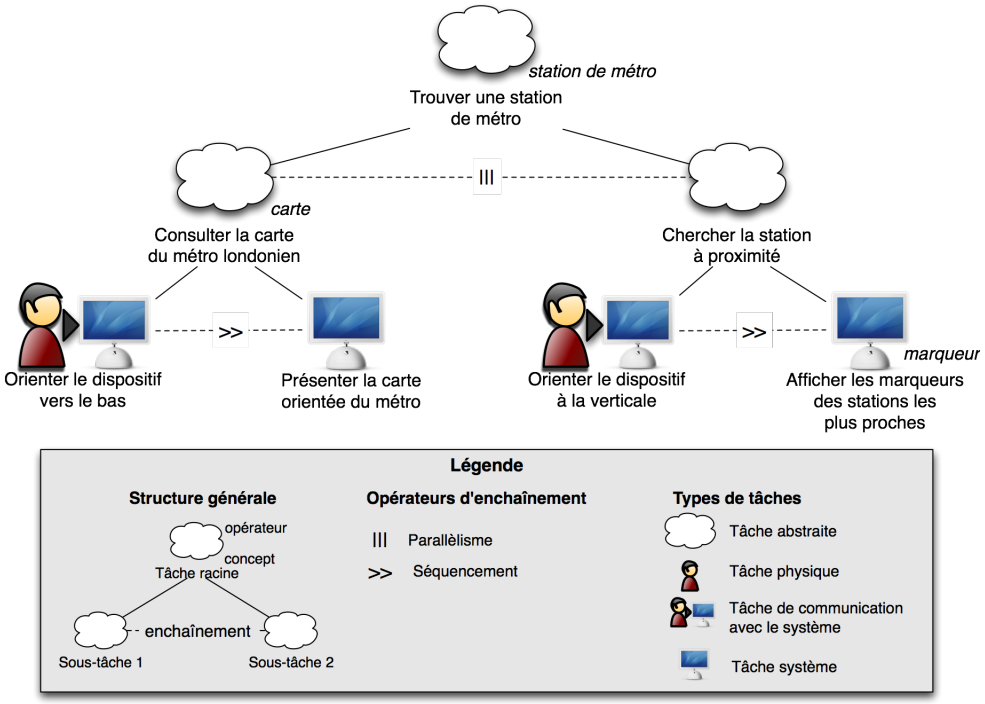


Figure 1.4 – Diagramme CTT simplifié de l'application *Nearest Tube©*

Bien qu'adaptée à la description des interfaces classiques, les modèles de tâches ne permettent pas de rendre compte des dispositifs sous-tendant une interaction en réalité mixte ou de l'organisation des flux de données entre dispositifs, éléments virtuels et utilisateur. Il est donc nécessaire de recourir à d'autres types de notations, spécifiques au domaine de la réalité mixte.

1.2.3 Description des systèmes de réalité mixte

Les notations spécifiques aux SRM traitent généralement du problème de l'organisation des dispositifs et des flux de données (c.f. section 1.1.3). Les notations les plus courantes comptent les langages ASUR [46], IRVO [22] ou le modèle de l'interaction mixte de C. Coutrix [36]. Ces trois notations disposent d'un pouvoir expressif comparable. Les deux premières s'appuient sur des représentations graphiques proches, tandis que le modèle de l'interaction mixte de C. Coutrix a recours à la métaphore de l'interaction instrumentale de M. Beaudoin-Lafon [8].

Nous utilisons, à titre illustratif, la notation ASUR, qui a pour avantage de s'appuyer sur un méta-modèle explicite [48], facilitant ainsi son instrumentation.

Les diagrammes ASUR décrivent l'organisation des dispositifs dans le contexte d'une technique d'interaction, et sont constitués de deux types d'entités : les composants et les relations entre composants.

1.2.3.1 Composants ASUR

Les composants ASUR contiennent la sémantique nécessaire à la conception et au développement des systèmes mixtes. Les quatre types de composants sont : Entité Réelle, Système, Utilisateur et Adaptateur :

- les **entités réelles** ou **personnes**, sous forme d'**outils réels** aident à la réalisation de la tâche ; les **objets réels**, participent à l'interaction en tant que cible de la tâche en cours ;
- le **système** peut être détaillé en termes d'**outils numériques** modifiant le comportement ou l'apparence d'autres entités numériques, d'objets représentant des concepts du domaine. Ces derniers peuvent eux-mêmes se décliner en **objets numériques** du domaine cibles de la tâche en cours ou en **propriétés numériques** liés à la tâche et servant de support à l'interaction (par exemple le retour d'information représentant un pointeur) ;
- l'**utilisateur** identifie l'un des utilisateurs du système ;
- les **adaptateurs en entrée** ou **en sortie** correspondent à des traductions des données du monde physique vers le monde numérique (par exemple, la capture d'un signal sonore à l'aide d'un microphone) ou du monde numérique vers le monde physique (par exemple, la représentation d'une donnée à l'aide d'un écran).

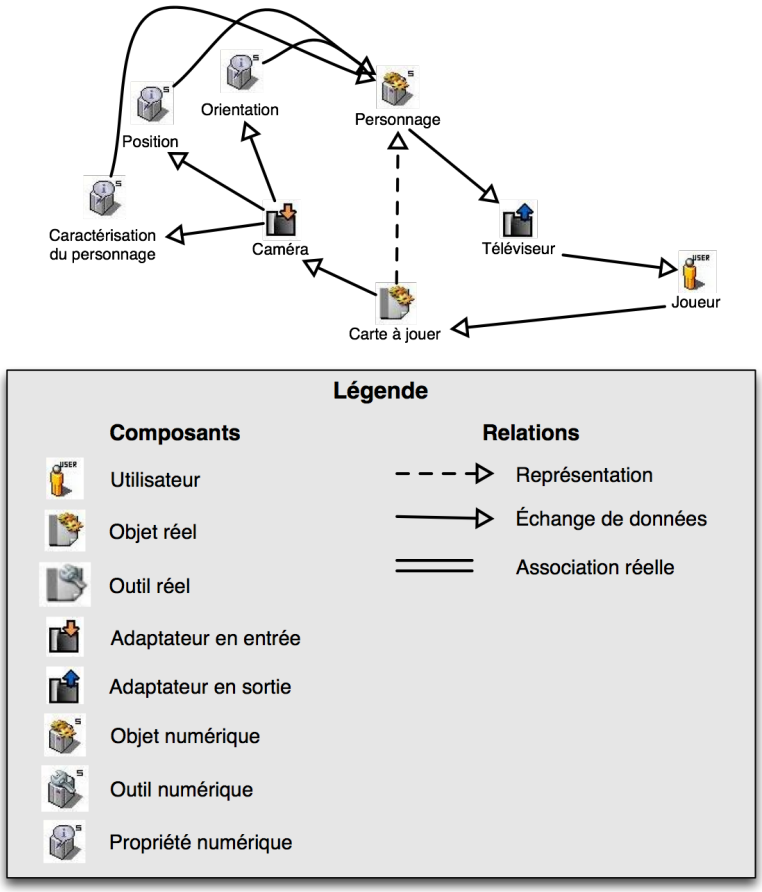
1.2.3.2 Relations

Le formalisme ASUR permet de relier des composants selon quatre types de relations :

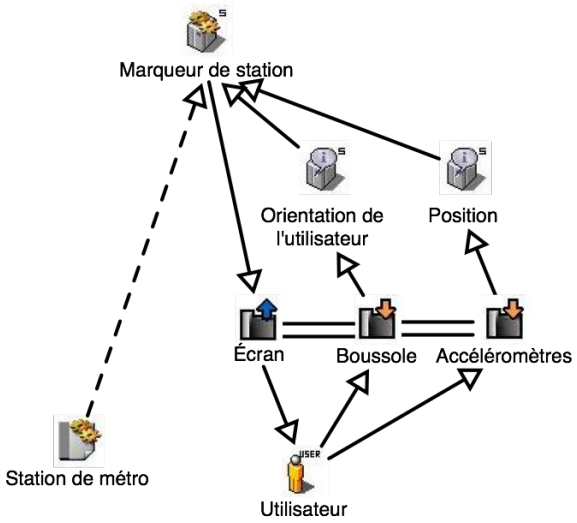
- **échange de données (A ->B)** : cette relation est unidirectionnelle et s'effectue selon un langage d'interaction donné et une indication quant aux dimensions nécessaires pour représenter les données, l'une ou l'autre donnée pouvant être spécifiée sur les diagrammes ;
- **causalité (C=>D)** : cette relation matérialise l'échange de données entre deux composants A et B, tel qu'on l'a détaillé plus haut, conditionné par une contrainte entre deux composants C et D ;
- **association réelle (Real association : A=B)** : cette relation représente une proximité physique persistante entre deux entités, qui peut être matérialisée par une superposition, une colocalisation etc. ;
- **représentation (A- ->B)** : Cette relation exprime la représentation d'une entité physique A par une entité numérique B, ou inversement selon l'objet physique ou numérique de la tâche. Elle peut être dynamique.

Le diagramme ASUR présenté en figure 1.5(a) retranscrit l'interaction homme-machine au centre du jeu *Eye of Judgement*© : le joueur manipule des cartes à jouer (l'objet physique de la tâche) identifiées par la caméra (adaptateur en entrée), dont on omet par concision la capture des codes-barres bidimensionnels. De cette capture sont extraites trois données : la position et l'orientation de la carte, ainsi que le personnage caractérisé par le code-barre bidimensionnel (données devenues des informations virtuelles). Ces données sont ensuite utilisées par le système pour représenter un personnage virtuel (objet virtuel de la tâche), affiché sur le téléviseur (adaptateur en sortie).

Le diagramme ASUR illustré en figure 1.5(b) correspond à l'interaction homme-machine de l'application *Nearest Tube*©. Ce diagramme nous permet de constater les différences fondamentales entre les deux conceptions. Ainsi, contrairement à l'exemple précédent, l'utilisateur interagit directement avec un dispositif (le couplage de l'écran avec boussole et



(a) Application *Eye of Judgement*©



(b) Application *Nearest Tube*©

FIGURE 1.5 – Diagrammes ASUR simplifiés

les accéléromètres) au lieu de l'objet identifié par le système du premier exemple. D'autre part, l'objet réel de la tâche (la station de métro) n'intervient pas directement dans l'interaction : l'utilisateur n'en a la perception qu'au travers de sa représentation numérique : le marqueur.

De même que pour les modèles, différents processus documentent la conception des interfaces homme-machine. Nous en abordons quelques-uns dans la section suivante.

1.3 Processus de développement pour l'IHM et les systèmes de réalité mixte

Nous traitons dans cette sous-section les processus de conception des interfaces classiques et des systèmes de réalité mixte. Nous soulignons dans les prochains paragraphes les concepts essentiels de ces démarches, ainsi que les correspondances sémantiques qu'elles entretiennent, et sur lesquelles nous nous appuyons pour nos propres travaux.

1.3.1 Méthodologies de conception des IHM classiques

Nous présentons ci-dessous trois approches de la conception des interfaces classique : la conception centrée utilisateur, issue de domaine de l'ergonomie logicielle, la conception centrée usage, plus proche du domaine du génie logiciel, et enfin le processus CAMELEON-RT, produit au sein de la communauté de l'interaction homme-machine.

1.3.1.1 La conception centrée utilisateur (ISO 13407)

À la suite d'une première norme (ISO 9241) axée sur la qualité ergonomique finale du produit, la norme ISO 13407, proposée en 1999, met en œuvre un processus de conception complet ayant vocation à être intégré dans un cycle de développement logiciel « classique ».

L'hypothèse sur laquelle s'appuie cette norme suggère que utilisateurs finaux sont les mieux placés pour déterminer et valider leurs besoins applicatifs. La conception centrée utilisateur impose donc un développement guidé par l'utilisateur, plutôt que par les possibilités technologiques ou l'expertise des membres de l'équipe de développement dévolus à la conception de l'interface.

La norme ISO 13407 définit cinq principes essentiels pour la conception centrée utilisateur :

- une prise en compte des utilisateurs, de leurs tâches et de leur environnement en amont du cycle de développement ;
- la participation active des utilisateurs dans la conception, à la fois pour l'évaluation des solutions d'interaction et pour l'identification de leurs besoins et des exigences liées à leurs tâches ;
- une répartition appropriée des fonctions entre les utilisateurs et la technologie ;
- une démarche de conception itérative ;
- une équipe de conception multi-disciplinaire, impliquant l'ensemble des acteurs impliqués dans la conception du produit final, issus des facteurs humains, du marketing, de l'assurance qualité etc.

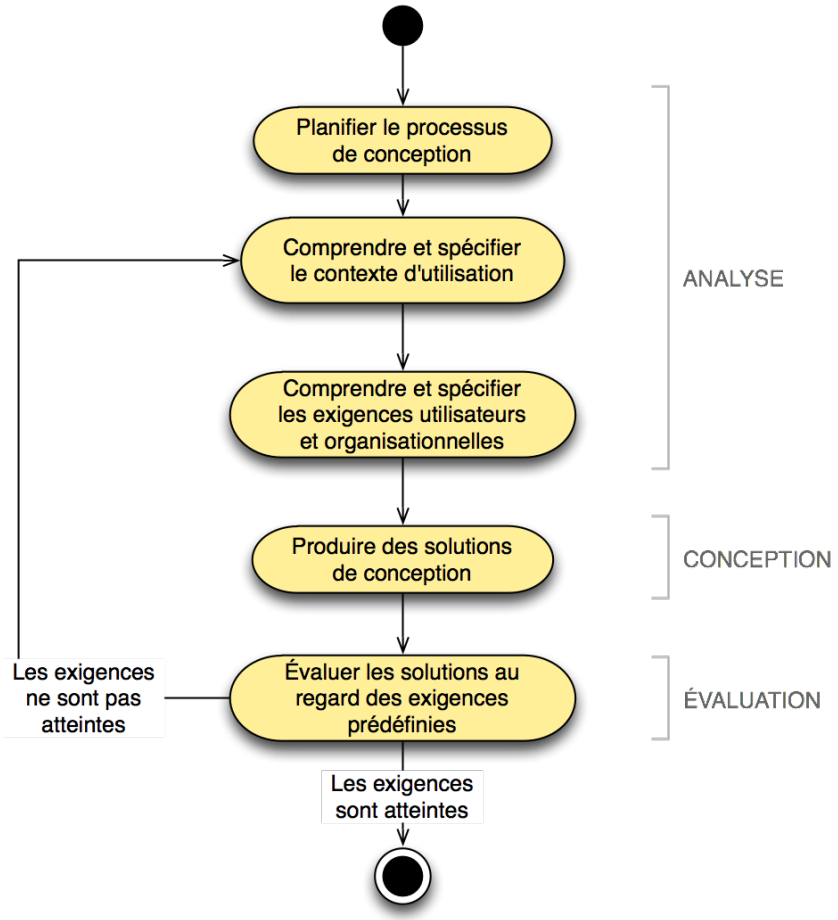


Figure 1.6 – Processus de conception centrée utilisateur (norme ISO 13407)

Ces cinq principes sont concrétisés dans une démarche composée de trois phases principales : l'**analyse**, la **conception** et l'**évaluation**. Celles-ci sont reprises en figure 1.6. Les détails du sous-processus d'analyse y sont également présentés.

En 2002, un rapport technique a précisé les aspects méthodologiques de la norme ISO 13407 en décrivant douze méthodes génériques pouvant être appliquées à différents instants du cycle de développement du logiciel, en fonction des caractéristiques du projet, des affinités des ergonomes, de la disponibilité des utilisateurs ainsi que des marges temporelles et financières. Ces techniques, dont le détail n'est pas le sujet de cette étude, comprennent entre autres les entretiens, les focus groups, l'observation des utilisateurs en situation de travail, des techniques issues du domaine de l'ethnographie etc. Cette démarche de conception impose donc peu de contraintes à l'ergonome, qui en adapte la mise en œuvre en fonction de son expertise personnelle. Nous présentons dans les paragraphes suivants les objectifs de chaque activité du processus de conception centrée utilisateur.

Planifier le processus de conception centrée utilisateur

Cette première activité d'analyse remplit d'abord un rôle pédagogique auprès de l'équipe de développement : l'ensemble des intervenants doivent s'entendre sur la valeur apportée par

la conception centrée utilisateur au développement.

D'autre part, les exigences du système sont clairement explicitées. Les ergonomes doivent également envisager la manière la plus opportune d'intégrer les préconisations de la norme ISO 13407 dans le cycle de conception logicielle, afin de faciliter l'atteinte des objectifs organisationnels.

Enfin, les ergonomes et l'équipe de développement s'accordent sur le type de conception adopté, ainsi que sur les méthodes de la conception centrée utilisateur utilisées et les différents niveaux d'implication attendus de la part des utilisateurs.

Comprendre et spécifier le contexte d'utilisation

L'objectif de cette activité est de recueillir une base de données concernant les environnements technique, physique, ambiant, social, organisationnel et législatif, ainsi que les contraintes matérielles (type de matériel informatique) et le ou les profils des utilisateurs finaux (connaissances, formation, compétences, fonctions, tâches à accomplir, caractéristiques physiques et psychologiques, habitudes etc.). Ce dernier point est bien évidemment central pour la suite de la conception, car elle permet aux ergonomes de déterminer le type de méthode d'évaluation à adopter par la suite ainsi que de définir des groupes de test.

Comprendre et spécifier les exigences utilisateur et organisationnelles

Il s'agit pour cette activité de définir les besoins, compétences et l'environnement de travail pour l'ensemble des acteurs impliqués dans le futur système. Les buts et tâches des utilisateurs sont identifiés puis organisés ; ces derniers sont ensuite profilés et des cas d'utilisation détaillés sont décrits puis priorisés. Ces cas d'utilisation incluent, entre autres, des objectifs d'utilisabilité mesurables tels que le taux d'erreur ou le rythme d'apprentissage.

Cette activité intègre également les exigences organisationnelles en termes de processus et de flux d'échanges (tel que le taux de dossiers traités d'un centre de service après-vente).

Produire des solutions de conception

Cette étape de la conception vise à intégrer l'ensemble des données accumulées lors des activités précédentes pour concrétiser des solutions d'interaction, qui seront modifiées en fonction des évaluations utilisateur. Il s'agit notamment pour l'ergonome de proposer un éventail de choix de conception : charte graphique, type d'interaction, choix de navigation. L'ensemble de ces choix sont intégrés dans des prototypes, pouvant aller de maquettes papier, présentations interactives jusqu'à des implémentations au noyau fonctionnel simplifié.

Évaluer les solutions au regard des exigences prédéfinies

Les prototypes réalisés précédemment sont utilisés pour entreprendre les évaluations ergonomique, en fonction des exigences définies en amont du cycle de conception. L'ergonome définit alors un protocole d'évaluation comprenant une ou plusieurs techniques d'évaluation permettant de recueillir la satisfaction des utilisateurs concernant la solution développée. Le

cas échéant, il pourra être décidé de réaliser une nouvelle itération de conception intégrant les remarques des utilisateurs.

1.3.1.2 La conception centrée usage

La conception centrée usage est une approche de conception d'interfaces utilisateur dirigée par les modèles, issue des travaux de L. Constantine et L. A. D. Lockwood [29] et promue par D. N. Norman [82]. Elle prend le contre-pied de plusieurs principes fondamentaux de la conception centrée utilisateur, notamment la focalisation de la conception sur le profilage sociologique, psychologique, ethnologique, physiologique etc. des utilisateurs. Au-delà d'une profusion de données parfois inexploitable de par son volume, la conception centrée utilisateur ne propose pas de processus pour en déduire l'interface utilisateur. Constantine et Lockwood reprochent ainsi à cette approche de réduire la démarche de conception à une suite d'essais suivis de corrections répétées et d'ajustements opérés en collaboration avec les utilisateurs. Plus généralement, les principaux écueils du développement de l'IHM basé sur le prototypage itératif suivi d'un retour utilisateur sont, selon les auteurs [28] :

- Cette approche donne l'*« illusion de la progression du développement »*, en multipliant les prototypes papier à un point de la conception où l'analyse du système est encore embryonnaire,
- Elle incite les utilisateurs et les concepteurs à *« se concentrer sur des détails graphiques secondaires »*, tels que les interacteurs,
- Elle *« décourage l'audace et le courage des concepteurs »*, en facilitant l'abandon de leurs responsabilités (par exemple, la promotion d'une interaction estimée plus efficace) au profit de l'avis des utilisateurs,
- Elle *« s'appuie excessivement sur les utilisateurs »*, au risque de focaliser l'IHM de l'application sur les attentes des quelques utilisateurs interrogés.

Afin d'éviter ces écueils, la conception centrée usage s'appuie davantage sur les attentes fonctionnelles et les intentions de l'utilisateur et les responsabilités du système. Il s'agit de privilégier la performance de l'utilisateur dans la réalisation de la tâche, dont les auteurs estiment qu'elle s'accroît avec la durée d'utilisation du système et l'apprentissage de l'outil, si nécessaire au détriment de l'expérience utilisateur, généralement évaluée lors de la première approche du système par l'utilisateur. En effet, les auteurs considèrent que dans ce dernier cas, la qualité des interfaces est implicitement validée en fonction de la familiarité de l'utilisateur avec les solutions adoptées, indépendamment de leurs qualités ergonomiques.

Les attentes fonctionnelles de l'utilisateur sont formalisées sous forme de *cas d'utilisation essentiels*, correspondant aux tâches utilisateur et indépendamment des réactions du système. La figure 1.7 présente l'exemple d'un tel cas d'utilisation, dans le cadre d'un environnement d'exécution de tests logiciels. La partie gauche du cas d'utilisation essentiel illustre les intentions de l'utilisateur, selon une décomposition analogue à celle qui pourrait être réalisée à l'aide du formalisme CTT.

Cette formalisation des tâches utilisateur est ensuite traduite en espaces de travail et interacteurs abstraits, dont l'assemblage est alors soumis à la validation des utilisateurs. Un exemple d'interface abstraite, déduit du cas d'utilisation essentiel de la figure 1.7, est décrit en figure 1.8. On y retrouve notamment les responsabilités du système (montrer la liste des tests disponibles, montrer la configuration du test, indiquer les résultats ou le statut du test)

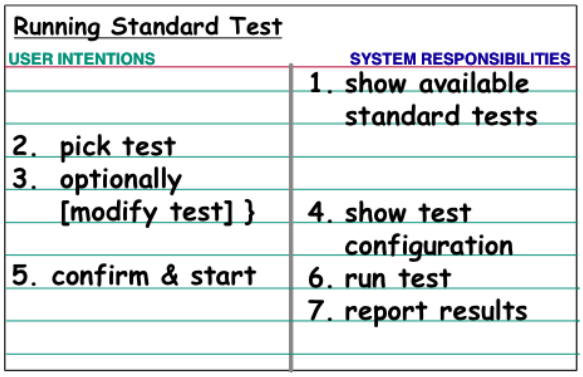


FIGURE 1.7 – Exemple de cas d'utilisation essentiel, d'après Constantine et Lockwood [29]

ainsi que des interacteurs abstraits permettant de réaliser les intentions de l'utilisateur : la liste des tests permet à l'utilisateur de réaliser et de modifier des sélections, et il est possible d'exécuter les tests sélectionnés.

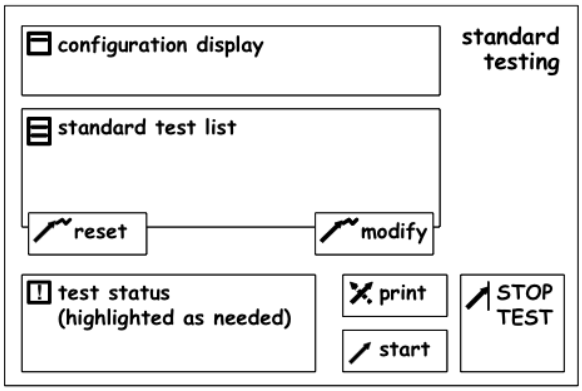


FIGURE 1.8 – Exemple de prototype abstrait, d'après Constantine et Lockwood [29]

L'évaluation ergonomique du système est ensuite réalisé en deux phases : des évaluations systématiques sont d'abord réalisées par des ergonomes. Les auteurs affirment que celles-ci permettent d'identifier une très grande majorité des failles ergonomiques plus rapidement qu'en ayant recours à des évaluations utilisateur. Par la suite, les utilisateurs sont finalement impliqués dans l'évaluation des prototypes. Ces expériences doivent permettre aux ergonomes d'estimer la progression des performances de l'utilisateur dans la réalisation de tâches fonctionnelles, au fil des utilisations.

Le tableau 1.2 résume les points de distinction essentiels entre conception centrée utilisateur et conception centrée usage.

1.3.1.3 Le processus de conception CAMELEON-RT

Bien que destiné à la conception des système interactifs *« distribués, migrables et plastiques* [3] *»*, il nous semble pertinent de décrire ici le processus issu du projet CAMELEON-RT [4]. En

3. *« Par analogie à la plasticité d'un matériau, la plasticité d'une IHM dénote sa capacité à s'adapter aux contraintes matérielles et environnementales dans le respect de son utilisabilité »* [118]

Tableau 1.2 – Différences fondamentales entre conception centrée utilisateur et conception centrée usage, d'après Constantine et Lockwood [29]

Conception centrée *utilisateur*	**Conception centrée *usage***
Focalisation sur l'utilisateur : expérience et satisfaction de l'utilisateur	Focalisation sur l'usage : amélioration des outils permettant la réalisation de tâches
Dirigée par les données des utilisateurs	Dirigée par les modèles
Implication substantielle de l'utilisateur : étude de l'utilisateur, conception participative, retour utilisateur et évaluations par les utilisateurs	Implication sélective de l'utilisateur : modélisation exploratoire, validation des modèles, évaluations structurées de l'ergonomie
Description des utilisateurs et de leurs caractéristiques	Modèles des relations de l'utilisateur avec le système
Modèles de conception réalistes ou descriptifs	Modèles de conception abstraits, permettant de retarder la prise en compte des détails concrets de l'interface
Conception par prototypage itératif	Conception par modélisation
Variété de processus informels ou non spécifiés	Processus systématique et spécifié

effet, on y retrouve mis en œuvre un ensemble de notions proches de la conception centrées utilisateur, telles que le profilage des utilisateurs, et de la conception centrée usage, telles que la description des tâches à haut niveau d'abstraction et la génération de modèles abstraits de l'interaction. Toutefois, dans le cadre de cette méthode, la génération des interfaces abstraites n'est pas destinée à l'évaluation, thème peu abordé dans le cadre du projet CAMELEON-RT, mais à la planification de la distribution de l'interface sur plusieurs plateformes matérielles : les différents espaces de travail sont répartis sur les différents dispositifs, puis les interacteurs concrets propres à chaque dispositif sont générés automatiquement.

Nous nous concentrons ici sur les facettes du processus pouvant être appliquées dans le cadre d'un développement d'interfaces non plastiques. Celles-ci correspondent à la démarche de conception des IHM telle qu'enseignée aux étudiants de Master Recherche de l'UFR-IMA de Grenoble [35]. Nous en résumons les étapes essentielles en figure 1.9. Par souci de clarté, nous n'avons pas indiqué les produits consommés par chaque activité : considérant qu'aucune contrainte n'est imposée, les concepteurs s'appuient sur l'ensemble des produits élaborés auparavant pour réaliser l'activité courante.

Les premières activités de la démarche se basent à la fois sur certains concepts de la conception centrée utilisateur (en particulier, les techniques de profilage de l'utilisateur) et sur les principes de la conception contextuée de Beyer et Holtzblatt [13]. Cette dernière approche consiste essentiellement à s'appuyer sur des données recueillies « sur le terrain » pour construire le modèle de l'utilisateur, décrire son activité puis en envisager l'amélioration, qui sera décrite sous forme de modèle abstraits, à l'instar des IHM abstraites de la conception centrée usage.

La démarche ne fixe pas systématiquement toutes les étapes de la conception. Pour certaines d'entre elles (par exemple, l'acquisition du modèle de l'utilisateur), un ensemble d'outils conceptuels sont proposés, à adopter en fonction du contexte de développement. Nous détaillons ci-dessous les objectifs des différentes activités de la méthode CAMELEON-RT :

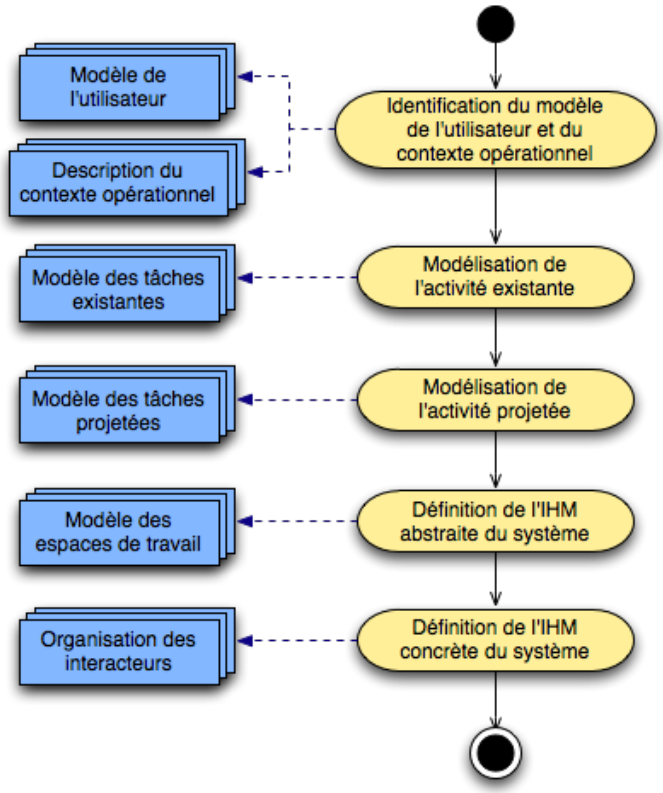


FIGURE 1.9 – Représentation du processus CAMELEON-RT adapté pour les interfaces non plastiques

Identification du modèle de l'utilisateur et du contexte opérationnel

Cette activité vise à comprendre l'utilisateur et son activité, dans un contexte opérationnel. Des entretiens avec les utilisateurs permettent de compléter cette compréhension. À titre d'illustration, nous identifions a posteriori un profil utilisateur adapté à l'application *Nearest Tube*© : cette dernière s'adresse aux voyageurs occasionnels visitant Londres, originaire des États-Unis ou du Royaume-Uni (utilisation des mesures en miles) ou du reste du monde, et peu familier avec les stations et lignes du métro londonien. L'application propose une interaction simple et intuitive, dont nous déduisons qu'elle vise aussi bien les utilisateurs familiers des nouvelles interactions que les utilisateurs moins expérimentés.

Modélisation de l'activité existante

Les tâches réalisées par l'utilisateur dans le contexte opérationnel existant sont étudiées. Une représentation abstraite, focalisée sur les tâches les plus pertinentes de son point de vue, est construite. Le formalisme CTT est généralement utilisé dans ce but.

Modélisation de l'activité projetée

L'équipe de développement envisage l'amélioration de l'activité existante des utilisateurs. Des entretiens avec ces derniers permettent de raffiner ces propositions, qui sont finale-

ment représentées à l'aide du formalisme CTT. Des cas d'utilisation UML, ainsi que des diagrammes de séquences, peuvent éventuellement compléter cette représentation. D'autre part, cette nouvelle vision du système est représentée sous forme de prototypes papier, par exemple des scénarimages (ou *storyboards*).

Définition de l'IHM abstraite du système

Cette activité vise à produire une structure de fonctionnement du système conforme aux attentes fonctionnelles des utilisateurs. Concrètement, une organisation de l'interface sous forme d'espaces de travail est déduite des modèles de tâches. Les liens de navigation entre ces différents espaces sont également représentés, ainsi que les concepts du domaine métier que ceux-ci manipulent. Un exemple d'IHM abstraite, déduit du modèle de tâches de l'application *Nearest Tube*© est présenté en figure 1.10.

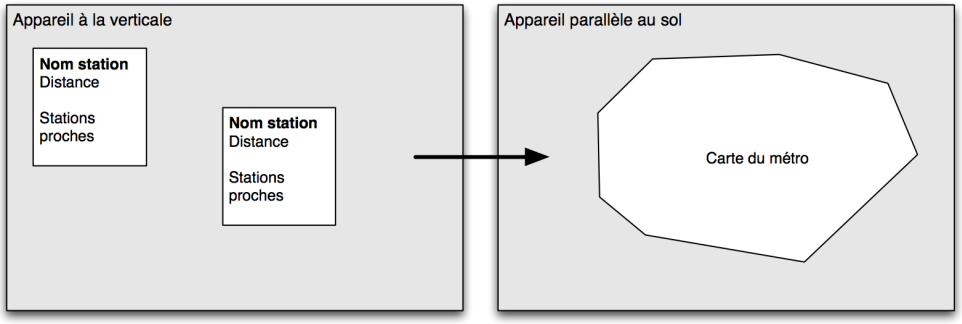


Figure 1.10 – IHM abstraite déduite de l'application *Nearest Tube*©

Définition de l'IHM concrète du système

Des techniques d'interaction concrètes sont déduites des liens de navigation, espaces de travail et concepts du métier décrits précédemment. L'équipe de développement organise ainsi les différents interacteurs qui seront implémentés dans l'IHM finale, c'est-à-dire celle exécutée par l'utilisateur.

1.3.2 Méthodologies de conception des systèmes de réalité mixte

Les processus de conception des systèmes de réalité mixte apparaissent comme des constructions distinctes des méthodes de développement d'interfaces homme-machine classiques. Nous présentons ci-dessus une approche intégrant les recommandations ergonomiques au centre de la conception des SRM, et une méthode centrée sur les scenarii.

1.3.2.1 Intégration de recommandations ergonomiques dans le processus de conception des SRM

Nous trouvons dans Charfi et al. [25] une approche originale de conception des systèmes de réalité mixte, illustrée en figure 1.11, basée sur l'insertion de recommandations ergonomiques (R.E) dans un processus de conception en trois étapes : définition des besoins utilisateur,

définition des tâches utilisateur et définition de l'interaction. Les deux premières étapes s'appuient sur la notation K-MAD, décrite par Charfi et al. [24] ; l'étape de définition de l'interaction utilise le modèle ASUR [46].

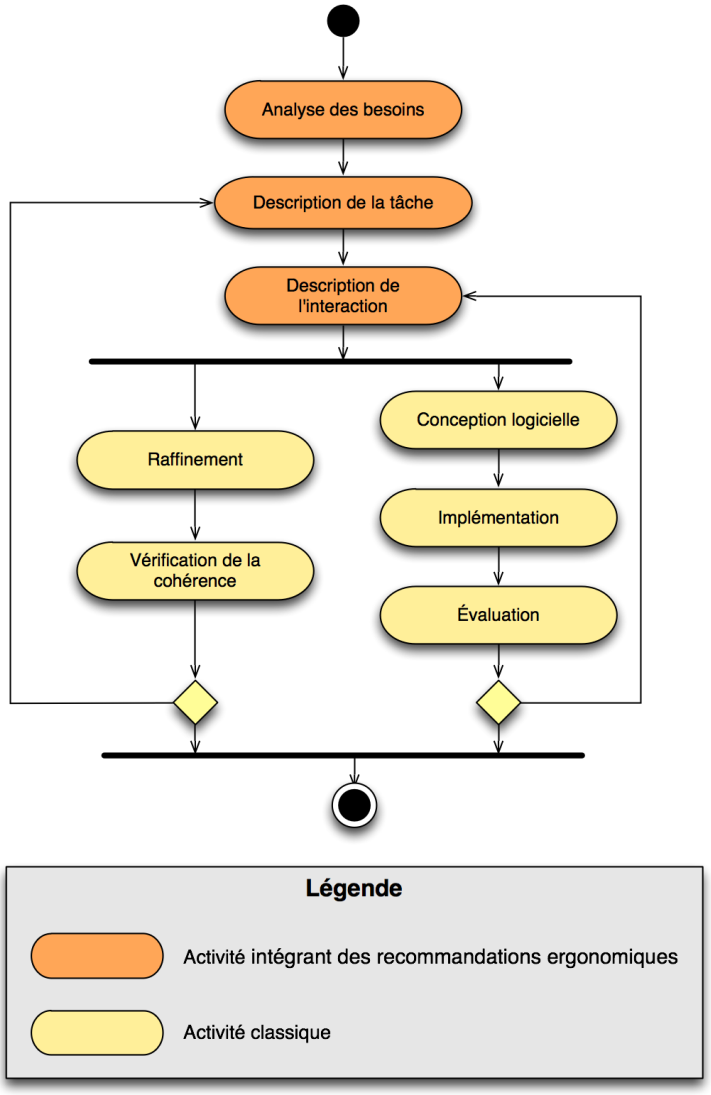


FIGURE 1.11 – Processus de conception des systèmes de réalité mixte, d'après Charfi et al. [25]

Les auteurs ont d'abord procédé à une veille scientifique portant sur les évaluations ergonomiques des systèmes de réalité mixte, ne conservant que celles décrivant *« les procédures, les mesures et les résultats »* [25]. Les recommandations ergonomiques tirées des études retenues ont été distinguées en recommandations générales valables pour toute interface, et recommandations spécifiques aux SRM [4]. Parmi ces dernières, les auteurs ont identifié les recommandations ergonomiques (R.E) applicables à la construction des modèles K-MAD et ASUR, ainsi que les étapes de conception auxquelles elles sont rattachées.

Les auteurs ont également distingué trois contributions des R.E :

4. Signalons que l'ensemble des recommandations ergonomiques est disponible en ligne à l'adresse suivante : `http://www.irit.fr/~Syrine.Charfi/RESIM`

- l'articulation des étapes du processus, par exemple *« dans une tâche de mémorisation d'objets numériques, privilégiant les préférences des utilisateurs et le taux de réussite, il est préférable de représenter les objets en 3D plutôt qu'en 2D »* (d'après Matthews et al. [77]) ;
- la mise en place d'un modèle, telle que *« le feed-back portant sur un objet doit être localisé à l'endroit où se trouve cet objet »* (d'après Chastine et al. [26]) ;
- l'articulation des étapes du processus combinée à la mise en place d'un modèle.

D'autre part, le découpage de la construction d'un modèle ASUR en 7 étapes essentielles permet aux auteurs de raffiner la catégorisation des R.E articulatoires entre les niveaux tâche et interaction en fonction de ces étapes (ces R.E articulatoires prennent alors pour cible un composant ASUR). Par exemple, la R.E *« dans une tâche de localisation d'une cible, privilégiant la performance, il est préférable d'utiliser le bird eye view plutôt que la perspective view pour le positionnement de cibles »*) spécifie la caractéristique « point de vue » d'une relation de transfert de données vers le monde physique. La conception s'organise ainsi en parcourant les recommandations ergonomiques applicables, pour l'étape de conception en cours de réalisation et en fonction du contexte d'interaction.

1.3.2.2 Le processus de conception des SRM collaboratifs et mobiles de P. Renevier

Les travaux de P. Renevier [95] abordent à la fois la conception des SRM, des systèmes collaboratifs et des systèmes mobiles. À cette fin, l'auteur propose un cadre d'intégration des différents modèles et fragments de processus non formalisés existants, sous la forme d'une démarche basée sur les scenarii, reprise ultérieurement par R. Chalon [20].

La démarche, synthétisée dans la figure 1.12, préconise de concevoir des scenarii basés sur l'analyse de la tâche réelle (observation par les ergonomes logiciel du travail de l'utilisateur et explication par ce dernier des phases pertinentes de son travail) et des scenarii provenant de l'analyse de l'activité (observation par les ergonomes logiciel et enregistrement vidéo de l'activité de l'utilisateur sur site, par exemple). Ces deux types de scenarii correspondent aux scenarii problème de Carroll (voir section 1.2.1). À partir de ces scenarii, un ensemble de besoins servant de base à la spécification du futur système est dérivé, puis fixé sous forme de scenarii projetés abstraits. Enfin, les fonctionnalités sont identifiées et intégrées dans des scenarii projetés concrets. Les techniques d'interaction sont conçues en fonction des spécifications des fonctionnalités lors de l'élaboration des scenarii projetés concrets. Enfin des évaluations sont réalisées dans les conditions d'utilisation attendues pour valider les propriétés fonctionnelles du système et son utilisabilité.

1.4 Synthèse

Ce chapitre présente les concepts et spécificités des systèmes de réalité mixte, lesquelles requièrent des modèles et pratiques de conception adaptées. À ce titre, nous avons dressé un panorama concis de plusieurs modèles et processus utilisés dans le cadre de la conception d'interfaces homme-machine classiques et des systèmes de réalité mixte. Bien qu'issus de domaines parfois différents, ces approches traitent de problématiques proches et partagent de nombreux points communs que nous ne manquerons pas d'exploiter dans nos contributions.

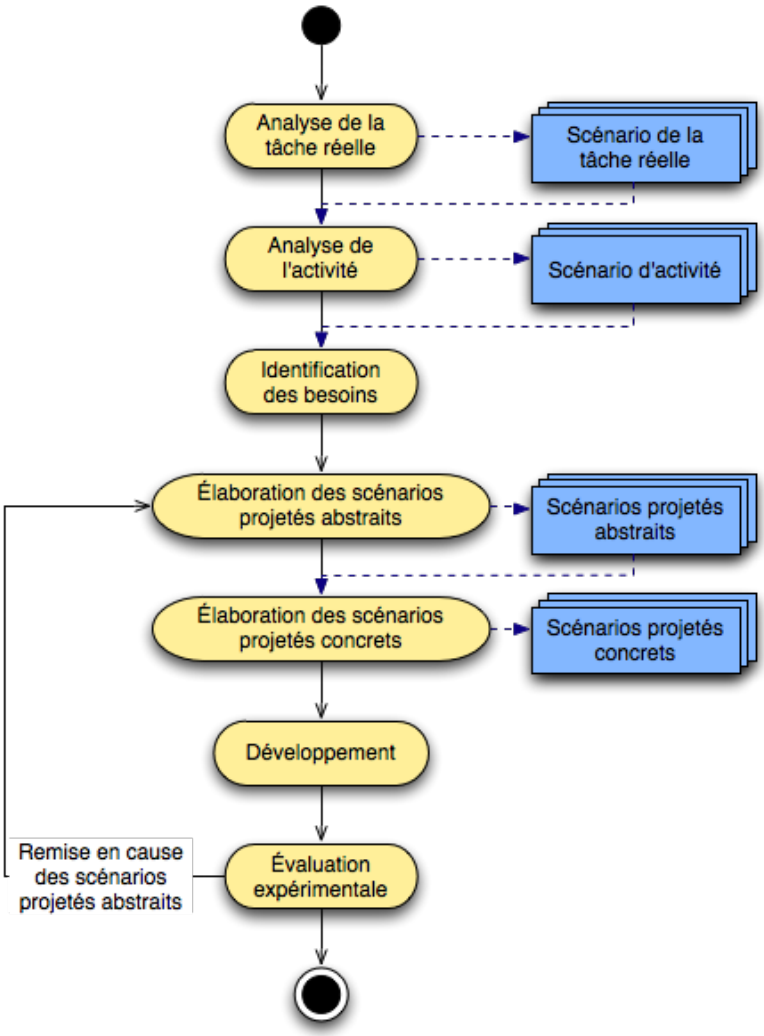


FIGURE 1.12 – Processus de développement pour les systèmes de réalité mixte collaboratifs, d'après P. Renevier [95]

*

* *

Auparavant, nous présentons dans le chapitre suivant une étude de plusieurs méthodes de développement issues de la littérature et intégrant des pratiques de conception du génie logiciel et de l'ingénierie de l'interaction homme-machine.

CHAPITRE 2

Méthodes de développement intégrant les pratiques du GL et de l'IHM

> "While I'm still confused and uncertain, it's on a much higher plane, d'you see, and at least I know I'm bewildered about the really fundamental and important facts of the universe." Treatle nodded. "I hadn't looked at it like that," he said, "But you're absolutely right. He's really pushed back the boundaries of ignorance."
>
> *Equal Rites*
> Terry Pratchett

Comme nous l'avons écrit précédemment, notre travail s'inscrit à l'intersection de deux domaines. Bien que souvent hermétiques l'un à l'autre, ces deux domaines n'ont pas évolué dans une ignorance réciproque. Par exemple, bien que peu de méthodes proposent un processus complet de développement de l'interface homme-machine, rares sont les travaux issus de la communauté GL ne traitant pas au moins a minima de notions d'IHM. Par exemple, la méthode IUT [60], élaborée par l'IUT d'Informatique de Grenoble, apporte à ses étudiants un processus GL complet et concis, et intègre dans son analyse de haut niveau des besoins une notation sommaire pour la description d'interfaces classiques. Inversement, la plupart des méthodes issues de la communauté IHM présentent des artefacts de développement GL, telle la méthode CAMELEON-RT [4] (c.f chapitre précédent), qui intègre une activité de description du modèle de données de l'application.

Toutefois, certaines approches de conception ont envisagé la conception des systèmes

comme une pleine intégration des préoccupations du génie logiciel et de l'interaction homme-machine. Nous nous intéressons dans ce chapitre à ces méthodologies, que nous analysons selon plusieurs catégories de critères.

2.1 Critères d'étude des méthodes GL/IHM

Nous décrivons dans les sections suivantes l'ensemble des critères que nous prendrons en considération pour réaliser une comparaison des différentes méthodes de développement GL/IHM. Nous avons ainsi opéré un découpage en trois critères généraux de caractérisation des méthodes et trois grilles d'analyse plus spécifiques portant sur les composants démarche, modèles et langages, outils et techniques.

En effet, selon D. Rieu [96], toute méthode est structurée selon quatre composants indissociables : une (des) démarche(s) (fil conducteur) guidant les activités de conception, les modèles représentant une vue du système, les langages (au travers desquels sont construits les modèles), et enfin des outils ou techniques permettant et facilitant l'utilisation et la mise en œuvre des démarches, modèles et langages. N'envisageant pas de traiter le niveau de formalisation des modèles utilisés dans les méthodes étudiées, nous considèrerons les composants modèle et langage simultanément.

2.1.1 Caractérisation des méthodologies

Suivant la caractérisation des méthodologies de Seligman et al. [104], nous abordons l'analyse des méthodes GL/IHM par l'identification des « philosophies » (ou *way of thinking*, selon Seligmann et al.) les soutenant. Dans le contexte de notre étude, nous classifions ces philosophies selon quatre propriétés fondamentales : les entités de premier ordre de la méthode, l'aspect du système sur lequel se focalise la conception, la caractérisation du cycle de développement et enfin la caractérisation des acteurs impliqués dans le développement. Ces propriétés constituent la base conceptuelle de construction des méthodes, et représentent par là-même des critères de choix fondamentaux pour orienter le développement du Système d'Information.

2.1.1.1 Entités de premier ordre

Les entités de premier ordre d'une méthode constituent les éléments de plus large granularité de découpage du système, et sont par conséquent parmi les premiers artefacts produits au cours du processus de développement. Ils définissent également, par extension, l'approche de découpage du système, l'organisation du développement, et le squelette sur lequel s'articulent l'ensemble des produits de la méthode. Les cas d'utilisation sont un exemple type d'entité de premier ordre : nous verrons dans la suite de ce chapitre qu'ils guident généralement un découpage par les fonctionnalités majeures du système.

2.1.1.2 Focalisation du développement

La focalisation définit l'aspect du système sur lequel se concentre en priorité la conception, conséquemment celui dont on peut attendre une qualité supérieure, ou bien la plus grande

valeur ajoutée, à l'issue du développement. On retrouve parmi les méthodes modernes des orientation variées, allant de l'architecture (par exemple, les composants) à l'utilisateur.

2.1.1.3 Caractérisation du cycle de développement

La caractérisation du type de développement spécifie l'enchaînement des activités de la méthode, considérant plus particulièrement :

- la répartition globale de l'effort de développement : les entités de premier ordre sont-elles développées séquentiellement (avec par exemple l'objectif de livrer l'application de manière incrémentale) ou parallèlement (l'application n'est livrée qu'une fois tous les blocs finalisés) ;
- la répartition locale de l'effort de développement, pour chaque entité de premier ordre : les activités sont-elles également réalisées de manière séquentielle, c'est-à-dire que les produits attendus en sortie de l'activité doivent être exhaustivement décrits avant d'aborder l'activité suivante, ou bien plusieurs activités peuvent-elles être réalisées en recouvrement ?
- l'itérativité de la méthode : une fois les activités finalisées, la méthode autorise-t-elle (et fournit-elle) des mécanismes de révision des produits ?

Ces différents paramètres sont généralement synthétisés dans une caractérisation plus globale : on parle ainsi de méthode « en cascade » pour désigner une méthode développant en parallèle les entités de premier ordre, de manière strictement séquentielle et sans révision possible [98]. Inversement, les méthodes agiles sont caractérisées par un développement globalement incrémental, localement parallèle et itératif [76].

2.1.2 Grille d'analyse des processus de conception

Nous décrivons dans cet intitulé les critères adoptés pour l'analyse des processus de conception, c'est-à-dire : les rôles fonctionnels impliqués dans le développement, les phases du développement couvertes par les méthodes étudiées, en prenant pour référence un découpage classique du cycle de développement, dont nous ne considérerons que les phases conceptuelles.

2.1.2.1 Rôles fonctionnels impliqués dans le développement

Les méthodes de développement mettent en jeu un nombre souvent important d'acteurs. À titre illustratif, la méthode RUP [72] fait référence à 31 acteurs distincts, dont la plupart sont liés au domaine Génie Logiciel, à l'expertise métier ou bien à l'ergonomie.

Or, en ce qui concerne la conception des systèmes mixtes, Dubois et al. [47] identifient quatre problématiques : l'identification des besoins et contraintes du domaine d'application, la conception logicielle, la conception de l'interaction et la conception et l'évaluation ergonomique. Nous reprenons cette classification dans notre analyse des méthodes de conception GL/IHM, à laquelle nous associons les quatres rôles fonctionnels suivantes :

- **spécialiste métier** ;
- **spécialiste GL** ;
- **spécialiste IHM** ;
- **ergonome**.

On notera que nous distinguons l'ergonome logiciel du spécialiste IHM de la manière suivante : nous considérons que le premier dispose d'une formation en psychologie sociale, anthropologie ou sociologie, ne possède pas de connaissances « fondamentales » en informatique, et axe sa pratique sur l'évaluation, le conseil, etc., en interaction homme-machine. Le spécialiste IHM, à l'inverse, est un informaticien spécialisé dans la conception d'IHM du point de vue logiciel, possédant des notions en ergonomie logicielle. Bien évidemment, dans la mesure où nous décrivons ici des rôles fonctionnels, il est envisageable qu'un acteur, initialement endossant le rôle de spécialiste IHM, puisse assumer également celui d'ergonome logiciel.

En effet, les différents acteurs intervenant effectivement dans le développement correspondent à des instanciations et spécialisations de ces rôles. Par exemple, un « analyste des processus métier » correspond à une spécialisation de « spécialiste métier ». On considère dans cette classification que les ergonomes logiciels possèdent les compétences nécessaires et suffisantes en psychologie cognitive, et que les spécialistes métier peuvent être des experts indépendants de la maîtrise d'ouvrage et de la maîtrise d'œuvre (consultants métier par exemple).

L'identification de ces quatre rôles fonctionnels du développement n'exclut pas l'implication dans la conception de la maîtrise d'ouvrage et des utilisateurs finaux (pour ces derniers dans le cadre de l'évaluation des prototypes, par exemple). Ce ne sont pas des rôles fonctionnels à proprement parler, dans la mesure où leurs préoccupations sont soit restituées sous forme de modèles intégrés au processus de développement, soit prises en charge par l'un des rôles fonctionnels (par exemple, l'ergonome dans le cas des résultats d'une concertation avec utilisateurs). Pour cette raison, nous n'étudions pas en détail leur implication dans les méthodes de conception, sauf exception.

2.1.2.2 Phases de développement traditionnelles

Comme nous le constaterons plus loin, les méthodes étudiées adoptent ou raffinent le découpage en phases classique des méthodes de développement GL : il s'agit en effet d'expliciter les besoins, puis de les analyser avant de réaliser une conception du système et son implémentation. Afin de comparer aisément les méthodes GL/IHM, nous utiliserons la terminologie suivante pour décrire les phases du développement :

1. **Analyse du métier** : un grand nombre de méthodes de développement propose, en préliminaire à la spécification du système, de décrire le métier de l'organisation commanditaire. Cette description peut consister en une énumération des profils d'utilisateur, une description concise ou bien exhaustive des activités manuelles et informatisées, telles qu'elles sont initialement pratiquées par l'organisation. Au-delà de familiariser l'équipe de développement avec les concepts et pratiques du métier en cours d'informatisation, cette phase permet souvent d'identifier des points de blocage (activités laborieuses, peu efficaces, redondantes etc.), dont la résolution peut apporter une valeur ajoutée conséquente au futur système.
2. **Spécification des besoins** : souvent réalisée en étroite collaboration avec les futurs utilisateurs, cette phase consiste en la description des besoins que le futur système devra satisfaire. Elle permet généralement d'identifier les concepts et processus applicatifs essentiels du futur système, à un haut niveau d'abstraction. Il devient possible, à

partir de cette description, de proposer un planning prévisionnel de développement.

3. **Analyse des besoins** : cette phase raffine les spécifications élaborées précédemment pour les traduire dans une solution fonctionnelle. On retrouve généralement à ce niveau du développement une transition depuis des descriptions utilisant des notations proches de la langue naturelle (Use Cases UML [87], scenarii) vers des notations plus proches du code (diagrammes de classes UML).
4. **Conception** : l'objectif de cette phase consiste à compléter la solution fonctionnelle décrite au cours de l'analyse d'une solution technique. Il s'agit le plus souvent d'intégrer les besoins non-fonctionnels du cahier des charges, tels que la sécurité ou la persistance des données. Les activités de cette phase sont occasionnellement réalisées simultanément à l'analyse.

Nous avons évalué au cours de nos travaux le processus « principal » des méthodes, c'est-à-dire l'enchaînement logique de traitement des fonctionnalités du futur système. Plusieurs méthodes proposent toutefois des processus « annexes » permettant de raffiner ponctuellement certains aspects du système, en-dehors de l'enchaînement d'activités « principal ». Par exemple, RUP [72] propose des processus transversaux centrés sur le pilotage, la configuration du processus principal, les tests fonctionnels. De même, Symphony [58] envisage l'analyse des aspects techniques du système en développement en parallèle de l'analyse fonctionnelle (on parle alors de « branches » du développement). Nous présentons ces processus annexes à titre indicatif, sans opérer de comparaison ou d'évaluation.

Ce type de découpage en phases des méthodes de développement est par exemple utilisé pour Unified Process [62] et RUP [72].

2.1.2.3 Couverture des phases

Nous utilisons la description des phases présentée ci-dessus comme référentiel. Dans les sections suivantes, nous analyserons, pour chaque méthode étudiée, les phases couvertes, les phases couvertes par le développement de l'IHM, les phases intégrant des collaborations. De plus, nous associons à chaque critère d'analyse les rôles fonctionnels impliqués dans les phases concernées.

- **phases couvertes par la méthode** : ce critère identifie les phases de notre référentiel couvertes par la méthode étudiée ;
- **phases couvertes par le développement de l'IHM** : ce critère détermine les phases du référentiel intégrant, pour la méthode étudiée, un processus (ou partie de processus) guidant le développement de l'IHM ou son évaluation ;
- **phases intégrant des collaborations** : au travers de ce critère, nous voulons évaluer les phases du référentiel auxquelles correspondent, pour la méthode étudiée, des collaborations explicites entre acteurs du développement. En particulier, il s'agit de repérer les parties du processus étudié mentionnant des collaborations, ou bien des modèles dont la réalisation nécessite la coopération de différents rôles fonctionnels, c'est-à-dire d'acteurs issus de cultures professionnelles différentes, nécessitant par conséquent des dispositions particulières pour permettre leur collaboration.

Le tableau 2.1 ci-dessous reprend l'organisation des différents critères de cette grille d'analyse. Pour chaque cellule du tableau, nous distinguons les rôles fonctionnels impliqués dans la phase traitée (voir la légende du tableau).

TABLEAU 2.1 – Grille d'analyse des processus

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	M | G | I | E	M | G | I | E	M | G | I | E
Spécification des besoins	M | G | I | E	M | G | I | E	M | G | I | E
Analyse	M | G | I | E	M | G | I | E	M | G | I | E
Conception	M | G | I | E	M | G | I | E	M | G | I | E

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome ; ND : Non déterminé

2.1.3 Grille d'analyse des modèles et langages

Dans tous les domaines de l'informatique, comme dans de nombreux domaines humains, l'utilisation de modèles et de langages comme abstractions de systèmes complexes est aussi systématique que nécessaire. Dans le cadre de l'étude des méthodes de développement GL/IHM, nous considérons en particulier les différents types de modèles rencontrés ainsi que leur(s) fonction(s). Nous distinguons ainsi quatre types de modèles :

- les **modèles « classiques » issus du GL**, en référence aux modèles utilisés communément dans l'industrie, les sociétés de service en informatique etc., dont les notations UML (Unified Modeling Language) [86, 87] et Merise [92] sont des exemples typiques. D'un point de vue historique, ces notations se sont imposées comme des langages de modélisation polyvalents, capables dans une certaine mesure d'exprimer la majeure partie des préoccupations liées au développement et au déploiement d'une application. Le rapport technique présentant l'infrastructure du langage UML exprime clairement cette ambition : *« l'objectif d'UML est de fournir aux architectes système, aux spécialistes du génie logiciel ainsi qu'aux développeurs logiciel des outils pour l'analyse, la conception et l'implémentation de systèmes logiciels et la modélisation du métier et des processus assimiliés »* [1] [86] ;
- *a contrario* des modèles décrits plus haut, les **modèles spécifiques** se focalisent sur un ensemble restreint de préoccupations. À titre d'exemple, le domaine de l'IHM dispose de plusieurs modèles spécifiques : les modèles CTT [93] et MAD [101] se focalisent sur la description des tâches utilisateur, les notations ASUR [46] et IRVO [22] sur les relations entre dispositifs physiques, utilisateur et entités numériques etc. ;
- les **scenarii en langue naturelle** s'appuient sur l'approche par scénario formalisée par J. M. Carroll [18]. Écrits en langue naturelle, les scenarii s'avèrent être un outil privilégié pour la collaboration entre spécialistes issus de domaines différents ainsi qu'entre utilisateurs et concepteurs [102]. À ce titre, ils sont considérés par A. Sutcliffe comme un point de convergence entre les domaines de l'IHM et du GL [114] ;
- les **prototypes**, qu'ils soient sous forme de fragments de code élaborés par le spécialiste GL, de simulations, Magiciens d'Oz, pleinement fonctionnels ou simples démonstrateurs de concepts. Nous étudions la pratique du prototypage qui, constitue, selon Sutcliffe, un autre pont entre l'IHM et le GL [114].

1. *"The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes."*

Nous distinguons, pour ces quatre types de modèles, trois usages essentiels dans le cadre des méthodes GL/IHM :

- la **modélisation des aspects métier ou GL**, c'est-à-dire les modèles exprimant les concepts, relations et la dynamique des domaines métier et fonctionnel ;
- la **modélisation des aspects ergonomie ou IHM**, c'est-à-dire les modèles permettant de décrire les problématiques liées à la conception des interfaces homme-machine, telles que la description des tâches utilisateur, des techniques d'interaction ou l'organisation des dispositifs support d'interaction ;
- la **modélisation des mises en correspondance**, c'est-à-dire les modèles pivot, dont la fonction est d'exprimer les relations entre espaces conceptuels espaces métier et interaction. La manière dont cette mise en relation est réalisée peut être directe (telle la traduction des tâches issues du modèle CTT [93] vers des cas d'utilisation UML [86] décrite dans [105]) ou indirecte (impliquant des correspondances à cardinalité multiple entre concepts, auquel cas il est nécessaire de définir des règles complexes de correspondance, tel le langage MoLIC [42, 38]).

Considérant que notre étude porte sur les méthodes intégrant pratiques du GL et de l'IHM, nous évaluons également les mécanismes de communication et de collaboration entre acteurs du développement.

Concernant les modèles communicationnels, il nous semble pertinent d'en vérifier l'origine (IHM ou GL). Ceux-ci doivent permettre :

- la **capture des besoins utilisateur**, sous une forme : 1) non ambiguë ; 2) pertinente pour les deux parties que sont les décideurs (*stakeholders*) et l'équipe de développement ;
- la **validation des incréments** implémentés du système par les décideurs.

Concernant les modèles collaborationnels, il s'agit d'identifier les types de modélisation permettant à des acteurs du développement issus de métiers différents de travailler de concert.

Le tableau 2.2 ci-dessous synthétise l'ensemble de ces critères : concernant les modèles GL et les modèles spécifiques, nous précisons le ou les langages utilisés ; les cellules marquées « Comm » ou « Collab » dénotent respectivement l'utilisation de modèles en tant que modèles communicationnels avec les décideurs et/ou les utilisateurs, et l'utilisation de modèles comme support de collaborations entre acteurs du développement (issus de métiers différents). Enfin, nous précisons le rôle des scenarii et prototypes lorsque ces derniers sont utilisés.

Tableau 2.2 – Grille d'analyse des modèles

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL			
Modèle IHM			
Scénario			
Prototype			

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

2.1.4 Grille d'analyse des outils et techniques

Dernier élément constitutif des méthodes de développement, les outils sont un aspect souvent délaissé malgré leur nécessité. Ainsi, comme nous le verrons plus loin, l'utilisation de la notation UML dans la plupart des méthodes modernes s'explique non seulement par sa constitution en standard, mais aussi par la kyrielle d'outils de conception assistée par ordinateur instrumentant cette notation, à des degrés de fidélité divers. L'utilisation d'outils de développement est ainsi un facteur majeur dans la réduction du temps de mise sur le marché du logiciel, dans sa maintenabilité et sa documentation. Dans le cadre de notre analyse des méthodes GL/IHM, nous nous focaliserons sur trois types d'outils intervenant lors du développement, dont nous recensons l'utilisation selon la grille d'analyse décrite au tableau 2.21 :

Outils documentant la méthode tels que les sites internet décrivant la méthode RUP. Ces outils ont pour finalité de permettre aux différents acteurs du développement d'identifier précisément les activités dont ils sont responsables, ainsi que les produits à fournir.

Outils opérationnalisant le cycle développement de la méthode, que ce soit son processus (par ex., des outils de gestion de workflow ou de planification) ou ses modèles (par ex., des outils de conception permettant la construction et/ou la génération de modèles et/ou de code source). Ils permettent également aux rôles de pilotage d'assurer un suivi précis du développement, en fonction des activités réalisées et des produits finalisés.

Outils opérationnalisant l'exécution des applications développées à l'aide de la méthode. Il s'agit ici de considérer les outils offerts comme des choix techniques privilégiés lorsque utilisés comme supports à l'exécution des éléments d'implémentation résultants de l'application de la méthode étudiée.

TABLEAU 2.3 – Grille d'analyse de l'instrumentation

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
• Outil 1 • Outil 2	• Outil 3	• Outil 4 • Outil 5

2.2 Analyse des méthodes GL/IHM

Les grilles d'analyse présentées ci-dessus sont appliquées sur cinq méthodes de conception GL/IHM, que nous présentons dans cette section. Nous utilisons comme méthode « témoin » le Rational Unified Process [72] d'IBM. Considérant que cette méthode correspond à l'instantiation de la méthode « théorique » Unified Process, il nous semble pertinent d'également décrire cette dernière. Outre UP et RUP, nous détaillons les méthodes ou fragments de méthodes suivants : DIANE+[116, 117, 115], UPi [109, 106, 107, 108, 110], User Engineering/Ovid [97], Wisdom [84] et UCD Agile [51]. Ces différentes méthodes, les langages de modélisation auxquels elles font appel et leurs relations sont résumés en figure 2.1.

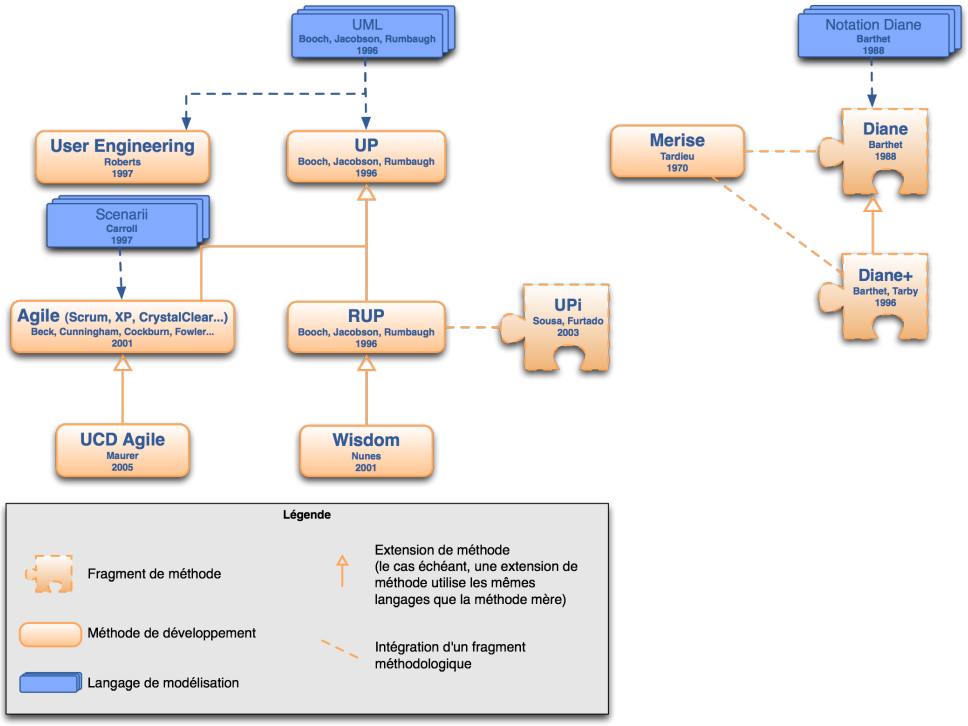


Figure 2.1 – Origines des méthodes comparées

On notera qu'aucune de ces méthodes ne propose à ce jour de processus de conception d'interfaces complexes. L'interface utilisateur de ces systèmes est même, pour la plupart, explicitement orientée vers des représentations sous forme de formulaires.

2.2.1 Le Unified Process

La méthode Unified Process (UP), décrite en 1999 par Jacobson et al. [62] est une base conceptuelle et abstraite sur laquelle s'appuient de nombreuses méthodes actuelles, proposée comme une rupture d'avec les précédentes approches de conception, telles que la méthode en V [94] ou les méthodes en cascade [98].

Unified Process s'appuie sur les propriétés suivantes :

- les entités de premier ordre sont les *cas d'utilisation* ;
- le développement est focalisé sur *l'architecture* et *la gestion du risque* ;
- le cycle de développement est *incrémental* au niveau global, *parallélisé* au niveau local, et *itératif*.

À ce niveau très général, l'utilisateur n'apparaît pas explicitement comme une préoccupation centrale du développement. Celui-ci n'y est d'ailleurs impliqué qu'au niveau de la description des cas d'utilisation, au cours de la phase de spécification des besoins.

La figure 2.2 décrit la structure de la méthode UP. Contrairement aux approches séquentielles [98], les différentes phases du développement (ici appelées *processus*) sont envisagées en termes d'un degré d'effort de développement (matérialisé dans la figure par les lignes épaisses courbes) pour chaque séquences de travail (appelées *phases* dans UP) pour

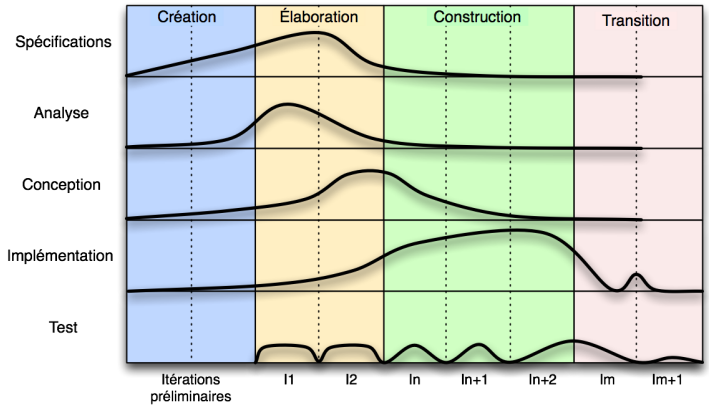


FIGURE 2.2 – Organisation de la méthode UP, d'après Jacobson et al. [62]

chaque phase. Le développement du système est ainsi organisé comme une évolution parallèle, organique, de l'ensemble des spécifications, de leur analyse, jusqu'au déploiement de l'application.

L'approche itérative de la méthode UP reprend les concepts introduits en 1986 par B.W Boehm dans son modèle de développement en spirale. Il s'agissait alors de décloisonner les phases du développement ainsi qu'il était pratiqué, dans la vision traditionnelle du modèle de développement en cascade [98] notamment. En effet, Boehm constate dans l'article *« A spiral model of software development and enhancement »* [14] qu'une approche séquentielle, non-itérative du développement, contraint les concepteurs à réaliser des phases de développement fonctionnellement correctes avant d'envisager la phase suivante. Il affirme que cet objectif est généralement irréalisable, en particulier lorsque l'équipe de développement ne maîtrise pas a priori le domaine applicatif. Son modèle de développement en spirale permet, au contraire, de reprendre aussi souvent que nécessaire les phases de développement en amont, dans une optique de raffinement progressif des spécifications jusqu'à l'implémentation.

En revanche, la méthode UP ne donne aucun détail sur la manière idoine de mener concrètement l'évolution d'une application. Il est seulement question de mener chaque itération à la manière d'un « mini-projet », résultante de la réalisation complète des différentes phases de la méthode. Par conséquent, le résultat attendu de chaque itération est une version partielle du produit final, ainsi qu'un ensemble de documents retraçant l'état actuel du développement.

2.2.2 Rational Unified Process (RUP)

Que cela soit imputé à son ancienneté (la première version de la méthode date de 1997), les nombreux outils l'instrumentant ou à sa large diffusion, le Rational Unified Process (RUP) est la méthode la plus exhaustive parmi celles analysées. Initialement élaborée par les concepteurs de la notation UML : Booch, Jacobson et Rumbaugh, cette méthode est une extension commerciale de la méthode théorique UP construite par les mêmes dans le cadre de la société Rational, rachetée en 2003 par IBM.

Le processus RUP est résumé dans la figure 2.3. Bien que les deux méthodes soient considé-

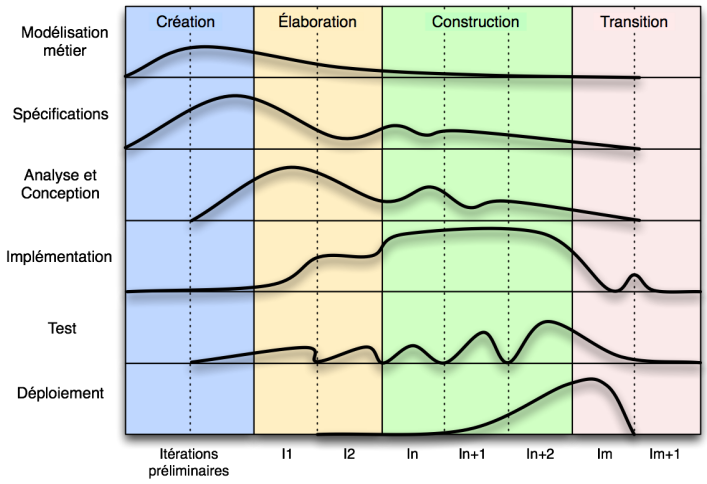


Figure 2.3 – Organisation de la méthode RUP, d'après Kruchten [72]

rées *« bien plus similaires qu'elles ne sont différentes »* [2], RUP étend ou redéfinit certains aspects de UP afin de rendre la méthode plus efficace dans une optique professionnelle : une phase de modélisation du métier permet notamment de mieux cerner les enjeux du système développé et la réalité du métier dans lequel celui-ci doit s'intégrer.

Outre la phase de modélisation du métier, on constate que la méthode fusionne les phases d'analyse et de conception ; elle introduit également une phase de déploiement.

On retrouve dans RUP l'approche de développement par séquences de travail introduit dans UP. La transition entre chaque séquence de travail est marquée par une « Étape essentielle » (*Milestone*), dont l'objectif est de valider le passage à la séquence de travail suivante Cette activité est réalisée par le « Relecteur de projet », sur la base des différents produits résultats de la séquence de travail courante. Par exemple, à l'issue de la séquence de création, le relecteur évalue les spécifications pour l'itération en cours, sa pertinence par rapport aux attentes des décideurs, les risques induits et le planning. En fonction de ces éléments, le relecteur de projet peut décider de faire reprendre cette séquence par les acteurs du développement concernés, ou bien d'annuler l'itération entière.

D'autre part, RUP propose des processus transversaux (dits « de support ») de pilotage (gestion de projet), configuration (gestion de l'environnement applicatif) et évaluation (campagnes de tests unitaires, d'intégration). Ce type de mécanisme permet notamment de reprendre certains produits de la conception afin d'y intégrer des résultats d'analyse *a posteriori*. Par exemple, le processus transversal de gestion de projet envisage d'évaluer à la fin de chaque itération les risques associés au développement du prochain cas d'utilisation.

Enfin, RUP est caractérisé par les mêmes propriétés que UP :
- les entités de premier ordre sont les *cas d'utilisation* ;
- le développement est focalisé sur *l'architecture* et *la gestion du risque* ;
- le cycle de développement est *incrémental* au niveau global, *parallélisé* au niveau local, et *itératif*.

Bien que la méthode soit dirigée par les cas d'utilisation, elle ne peut pas pour autant être considérée comme étant « centrée sur l'utilisateur ». En effet, du point de vue du RUP, les cas d'utilisation sont considérés comme *« une séquence d'actions réalisée par un système, qui*

produit un résultat observable apportant de la valeur à un acteur spécifique » [2] [72]. De même, une séquence d'actions est décrite comme *« un flot d'événements spécifique au travers du système »* [3] [72]. Ces définitions ne confirment effectivement que le centrage sur l'architecture de la méthode RUP.

Contrairement à UP, la méthode RUP documente différents types d'itération. En effet, différents « plans d'itération » sont proposés, en fonction de la maîtrise du domaine d'application, de la caractérisation de l'équipe de développement (expérience), de l'évaluation des risques liés au développement et des contraintes de mise sur le marché.

L'adoption d'un plan d'itération implique la planification d'une nouvelle itération de développement de l'application, dont le résultat attendu est une version exécutable du produit (interne ou externe au développement). De plus, chaque plan d'itération propose une organisation spécifique des séquences de travail (i.e. les « phases », au sens de RUP) : une séquence d'élaboration plus courte pour la construction d'un incrément dont le domaine d'application est bien maîtrisé, plus longue si le domaine est mal maîtrisé, par exemple.

2.2.2.1 Processus

Les caractéristiques du processus de la méthode RUP sont résumées dans le tableau 2.4. En particulier, nous notons que la conception de l'IHM n'intervient explicitement que durant la phase de spécification des besoins. Bien évidemment, l'IHM est implémentée ultérieurement, toutefois cette partie du système n'est pas distinguée du reste une fois celle-ci intégrée dans une preuve de concept (prototype) construite par l'architecte logiciel lors de la phase d'analyse et conception. Concrètement, la conception puis l'implémentation de l'IHM sont attribuées aux spécialistes GL.

TABLEAU 2.4 – Grille d'analyse du processus de la méthode RUP

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	M	–	–
Spécification des besoins	M|G	E	–
Analyse	G	–	–
Conception	G	–	–

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome

D'autre part, la méthode RUP n'intègre pas de collaboration explicite entre les acteurs du développement : les synthèses de produits (par ex., l'analyse des produits d'une séquence de travail lors d'une étape essentielle) sont réalisés par des acteurs tiers (par ex., le « Relecteur de projet ») et il n'existe pas de mécanisme de synchronisation entre les différents spécialistes. Par exemple, le prototype d'IHM réalisé par le « Concepteur de l'interface utilisateur »

2. *"A use case is a sequence of actions a system performs that yields an observable result of value to a particular actor"*

3. *"The sequence of actions referred to in the definition is a specific flow of events through the system"*

(correspondant au rôle fonctionnel de l'ergonome) est fourni pour intégration à l'« Analyste système », tel quel.

2.2.2.2 Modèles

Les caractéristiques des modèles employés dans la méthode RUP sont résumés dans le tableau 2.5.

De par son origine, nous constatons que la méthode fait principalement appel à l'ensemble des modèles UML pour exprimer les différentes facettes des systèmes, essentiellement orientées sur la description du métier et de l'architecture du système. Les cas d'utilisation sont considérés à la fois comme des entités de premier ordre et comme modèles pour la communication avec les décideurs et utilisateurs.

Enfin, comme nous l'avons vu plus haut, la conception de l'IHM est centrée sur la réalisation itérative de prototypes. Ceux-ci sont utilisés dans le cadre d'évaluations de l'ergonomie (i.e., de communication avec les utilisateurs), et sont également exploités par l'architecte logiciel comme une base de travail en vue de l'intégration ultérieure de l'interface homme-machine dans le système.

Tableau 2.5 – Grille d'analyse des modèles de la méthode RUP

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL classique	UML		
Modèle IHM			
Scénario	Cas d'utilisation{Comm}		
Prototype		Évaluation {Comm} et intégration {Collab}	

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

2.2.2.3 Outils

Le Rational Unified Process est instrumenté dès sa création, par ses concepteurs. L'un des premiers outils à voir le jour est un site Web documentant la méthode et permettant aux acteurs du développement de naviguer parmi les activités, objectifs et produits.

Une large suite d'outils opérationnalisant le développement est également proposée : outils de modélisation et de génération automatique de code (Rational Rose™, Rational Visual Modeler™), d'aide à la collaboration (Rational Team Concert™), de documentation automatique (Rational SoDA™), de gestion de projet (Rational ClearCase™) etc.

D'autre part, de nombreux outils alternatifs ont été proposés, faisant ainsi de RUP la méthode la plus instrumentée. L'ensemble de ces outils permet d'assister les différents acteurs du

développement à chacune des étapes de la conception, du recueil des besoins jusqu'au déploiement.

TABLEAU 2.6 – Grille d'analyse de l'instrumentation de la méthode RUP

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
• Description de la méthode sous forme de site Web	• Outils d'aide à la conception • Outils de génération de code • Outils de plannification/ pilotage	–

2.2.3 Unified Process for interactive systems (UPi)

Parmi les méthodes intégrant GL et IHM, UPi (*Unified Process for interactive systems*) combine deux approches originales : le recours à des principes issus du domaine de la recherche opérationnelle (RO) et sa construction comme un fragment de démarche, destiné à la conception de systèmes interactifs et pouvant être combiné à d'autres démarches, tel RUP. À ce titre, UPi ne décrit aucun mécanisme de pilotage des projets de développement, et limite la description de l'architecture interne de l'application aux fonctionnalités directement liées à l'interaction.

L'essentiel de la méthode reprend les principes de conception de la méthode RUP et de la conception centrée usage de Constantine et al. [29]. Nous ne considérons dans notre étude que les premières phases du processus UPi, c'est-à-dire la « collecte des besoins utilisateur » et l'« analyse et la conception ». Les phases ultérieures, traitant de l'implémentation et de l'évaluation, sortent du cadre de nos travaux.

De plus, la méthode UPi se veut particulièrement adaptée pour la conception d'interfaces pour lesquelles il existe peu de recommandations ergonomiques ou de pratiques de développement éprouvées. L'auteur de la méthode utilise ainsi comme exemple récurrent la conception d'un système de télévision digitale permettant à l'utilisateur d'interagir avec du contenu multimédia [108, 110].

Les premières activités de conception (c.f figure 2.4) sont focalisées sur le recensement des besoins fonctionnels et non-fonctionnels (activité « Éliciter les besoins des décideurs »), découpés en cas d'utilisation (activité « Découvrir les acteurs et les cas d'utilisation »). Chaque cas d'utilisation est alors analysé (activités « Détailler un cas d'utilisation », « Définir l'architecture » et « Appliquer le plan de définition de l'interface »), avant la production de prototypes papier soumis à validation (activité « Prototyper l'interface »).

Les produits de l'élicitation des besoins utilisateurs sont essentiellement textuels et recensent les profils utilisateur ainsi que les besoins fonctionnels et non-fonctionnels, sous forme de listes priorisées.

Le processus de raffinement des cas d'utilisation en modèles de tâches fait appel au formalisme CTT [93]. De plus, l'auteur préconise de transcrire les besoins reliés à des contraintes d'utilisabilité en « tâches d'utilisabilité ». Ces dernières s'appuient sur des « patrons d'utilisabilité », décrits dans de nombreux travaux [16, 119, 121, 54].

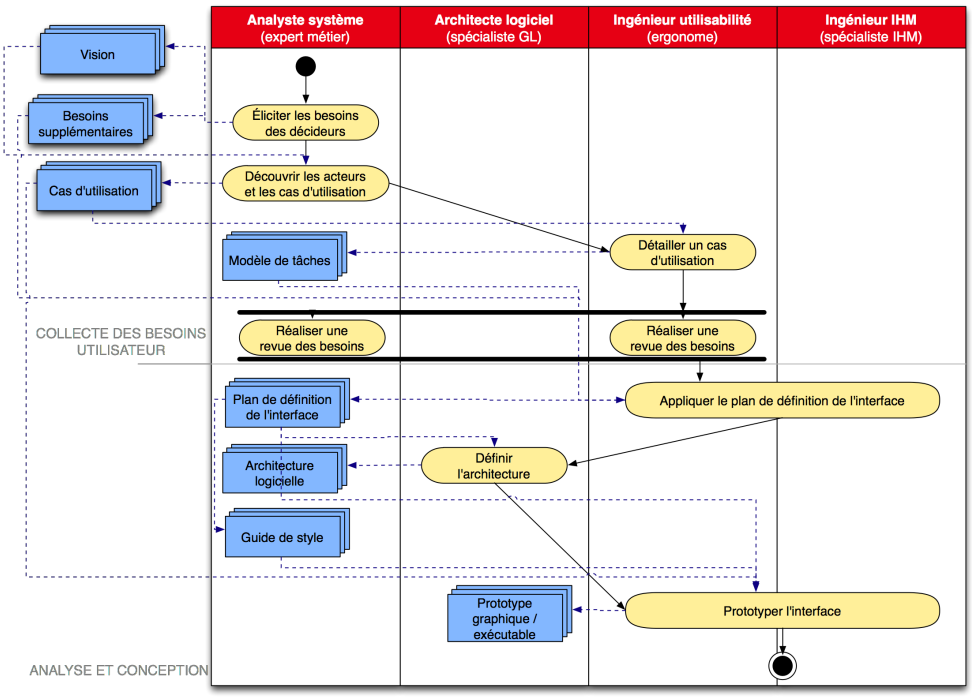


Figure 2.4 – Phases amont de la méthode UPi, d'après Sousa [110]

Les patrons d'utilisabilité consistent généralement en des techniques d'interaction répondant à une problématique ergonomique, telle la technique dite du « fil d'Ariane » (en anglais, *breadcrumb*), consistant à signaler la position de l'utilisateur dans un document structuré hiérarchiquement en indiquant la succession de niveaux hiérarchiques séparant ledit utilisateur de la racine du document. Cette technique répond à la contrainte ergonomique de guidage, qui précise qu'un utilisateur doit pouvoir percevoir sa localisation au sein d'une interface [7].

Puis, l'analyse des cas d'utilisation considère les besoins fonctionnels et non-fonctionnels pour déterminer les « patrons d'utilisabilité » les plus adéquats pour réaliser l'interface homme-machine.

À l'issue de la phase de collecte des besoins, une revue des produits de la phase est réalisée, éventuellement concrétisée par des prototypes papier fournis aux utilisateurs à des fins d'évaluation. Le cas échéant, une nouvelle itération de la phase peut avoir lieu, afin de corriger les produits jugés inadéquats.

La phase d'analyse et conception est marquée par la construction du « Plan de définition de l'interface » (*UI Definition Plan*). Cet artefact permet notamment aux concepteurs de faire face aux situations où le système développé utilise un type d'interaction pour lequel il n'existe pas de recommandations. Sa réalisation s'appuie sur l'outil MACBETH [4] d'aide à la décision, issu du domaine de la recherche opérationnelle, utilisé de façon à faciliter la sélection des patrons d'utilisabilité.

La construction du « Plan de définition de l'interface », s'opère en quatre étapes :

1. **Structuration** : Les différentes options suggérées par les concepteurs (ergonomes

4. http://www.m-macbeth.com

et spécialistes IHM) pour chaque problème d'interaction spécifique sont d'abord exprimées sous la forme de patrons d'utilisabilité. Les concepteurs définissent également un ensemble de critères (coût d'implémentation, d'efficience, de facilité d'apprentissage, etc.) qualifiant l'interface du futur système, ainsi que des valeurs d'évaluation de ces critères. Les options et critères sont intégrés dans l'outil MACBETH.

2. **Évaluation** : Chaque patron est évalué par les concepteurs suivant les critères sélectionnés, et obtient ainsi un score permettant à l'outil MACBETH de classer les différentes solutions d'interaction.
3. **Pondération** : Indépendamment des évaluations effectuées par les concepteurs, les utilisateurs finaux pondèrent les critères définis par ces derniers, une nouvelle fois à l'aide de l'outil MACBETH.
4. **Analyse** : Les évaluations des patrons d'utilisabilité sont pondérées selon les valeurs déterminées par les utilisateurs. Les valeurs résultantes sont alors consolidées pour générer un nouveau classement des patrons d'utilisabilité.

Les étapes d'évaluation, de pondération et d'analyse sont suivies d'une concertation entre acteurs du développement (concepteurs et utilisateurs finaux), focalisée sur les raisons motivant les choix de chaque option majoritaire.

Une fois les patrons d'utilisabilité choisis, l'architecture logicielle permettant la mise en œuvre des solutions d'interaction est décrite, à l'aide du langage UML. Cette approche fait référence aux patrons de mécanismes logiciels tels qu'initialement décrits par L. Bass [5] [6] : par exemple, s'il s'agit de mettre en œuvre un patron d'utilisabilité ayant recours à un mécanisme d'annulation (*Undo*), il est préconisé d'intégrer, en concomitance avec le mécanisme interactionnel, le composant fonctionnel supportant l'annulation. L'ensemble des patrons d'utilisabilité choisis lors de l'activité « Appliquer le plan de définition de l'interface » définit un squelette, autour duquel est bâtie l'interface complète.

La dernière activité de la phase d'analyse et conception guide le prototypage de l'interface homme-machine. Les types de prototypes préconisés regroupent tous les avatars « visuels » (maquettes papier, présentations animées, prototypes exécutables) de la solution de conception, auxquels est rattaché le guide style de l'application. Prototypes et guide de style sont alors fournis pour validation aux décideurs [108].

Les caractéristique de la méthode UPi sont synthétisées comme suit :

- les entités de premier ordre sont les *cas d'utilisation* ;
- la méthode est centrée sur l'*usage* et l'*architecture*, au sens de la structure de patrons d'utilisabilité et de composants fonctionnels les supportant ;
- la méthode est *itérative* et *incrémentale* à un niveau global ; les phases du cycle de développement étant marquées par des activités de validation verrouillant le passage à la phase suivante, on peut estimer l'effort de développement local comme *séquentiel*.

2.2.3.1 Processus

La méthode UPi opère une distinction claire entre les différents rôles de la conception (c.f figure 2.4), aisément mis en correspondance avec les rôles fonctionnels de notre grille

5. Les développements plus récents des travaux de L. Bass [64] ont finalement supprimé les modèles UML des description de patrons, ces derniers ayant été jugés trop prescriptifs par les concepteurs chargés de les appliquer.

d'analyse (c.f tableau 2.7). De plus, la démarche étant explicitement ciblée sur la conception de l'IHM, le développement cette dernière est suivi tout le long du développement.

Tableau 2.7 – Grille d'analyse du processus de la méthode UPi

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	–	–	–
Spécification des besoins	M	E	–
Analyse	G	I | E	–
Conception	G	I | E	–

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome

2.2.3.2 Modèles

La grille présentée au tableau 2.8 laisse percevoir la relative disparité des modèles utilisés par UPi. Cas rare parmi les méthodes étudiées, les besoins des décideurs et utilisateurs ne sont pas formalisés sous une forme structurée, laissant aux prototypes le rôle de modèles communicationnels.

Tableau 2.8 – Grille d'analyse des modèles UPi

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL	UML		
Modèle IHM		Modèle de tâches(CTT)	
Scénario			
Prototype	Tous types de prototypes (prototypes papier et prototypes « visuels »){Comm}		

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

2.2.3.3 Outils

Deux outils sont proposés dans le contexte du fragment de démarche UPi [108] : l'outil MACBETH pour la construction de l'architecture des solutions d'interaction et l'outil CTTE [80], utilisé pour l'élaboration des diagrammes CTT [93]. Bien évidemment, UPi n'étant qu'un fragment de méthode destiné à être intégré à des méthodes supportant le développement des aspects GL de l'application, il n'est pas surprenant de ne trouver que deux outils préconisés par les auteurs.

TABLEAU 2.9 – Grille d'analyse de l'instrumentation de la méthode UPi

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
–	• MACBETH • CTTE	–

2.2.4 DIANE+

DIANE+est une méthode née dans les années 80, avec pour objectif de compenser certaines lacunes de la méthode Merise [92], notamment en terme de « prise en compte des caractéristiques de l'utilisateur et la conception de l'interface » [117]. DIANE+ aborde donc la conception depuis les problématiques de l'IHM.

Initialement proposée par M.-F. Barthet [5] sous le nom de DIANE, la méthode a évolué au fil de la collaboration de l'auteur avec J.-C. Tarby. D'une approche centrée sur l'analyse des besoins utilisateur, la spécification des tâches et la conception de l'IHM, elle s'est ainsi dotée d'une représentation des données et du contrôle applicatif (OPAC) [116], basée sur le formalisme PAC [34].

Après plusieurs reformulations de DIANE+[117, 115], les auteurs se sont finalement recentrés sur les objectifs initiaux de la méthode. À ce jour, la méthode peut donc être considérée comme un fragment, dont la vocation est de s'intégrer dans Merise (à noter que cette intégration est désormais moins revendiquée).

DIANE+ concentre son processus sur deux phases : la spécification des tâches du point de vue de l'utilisateur du système (UPV, ou « User Point of View »), et la spécification des tâches automatisées, réalisées par le système (SPV, ou « System Point of View »).

Enfin, bien que les premières versions de la méthode se soient adressées à l'ensemble des acteurs du développement, notamment grâce à une modélisation semi-formelle simple de l'IHM, l'usage a vu l'adoption de la méthode se concentrer sur les spécialistes de l'IHM et de l'ergonomie.

Pour résumer, les caractéristiques de DIANE+sont les suivantes :

- les entités de premier ordre sont les *cas d'utilisation* (au sens de tâches utilisateur à haut niveau d'abstraction) ;
- la méthode est focalisée (centrée) sur l'*utilisateur* ;
- le cycle de développement est *incrémental* au niveau global, et *itératif* ; nous n'avons pu déterminer le type de développement au niveau local.

La phase de description du point de vue utilisateur (voir figure 2.5) consiste pour l'essentiel à identifier les « tâches essentielles » de l'utilisateur, c'est-à-dire les tâches à haut niveau d'abstraction, comparables aux travaux sur les « cas d'utilisation essentiels » de Constantine [29]. Celles-ci sont ensuite détaillées sous forme de scenarii ou grâce à la constitution d'un corpus d'entretiens avec les futurs utilisateurs [115]. Enfin, l'enchaînement précis des tâches (de niveau de raffinement supérieur, mais abstraites du point de vue des modalités) est modélisé.

Au niveau SPV, l'ensemble des tâches (utilisateur et système) est concrétisé, en termes de modalités et en fonction du contexte d'interaction (profils utilisateur, environnement,

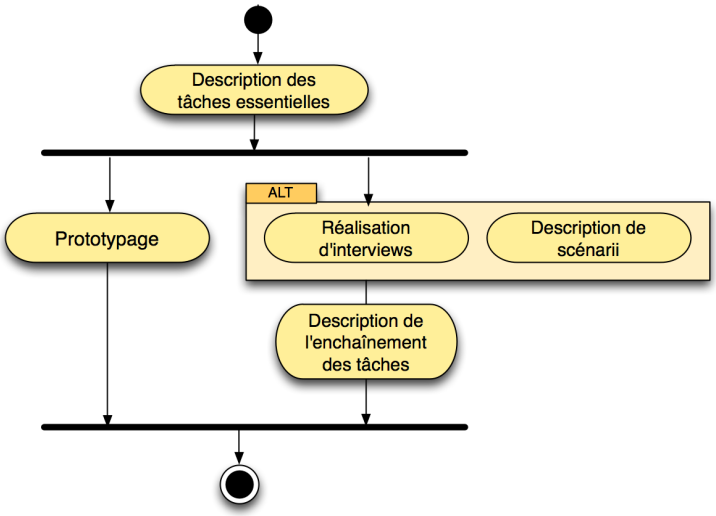


Figure 2.5 – Démarche de construction du UPV (User Point of View) de Diane+

plateformes matérielles).

Les itérations du cycle de développement sont déclenchées selon les résultats d'évaluation des prototypes, réalisés à l'issue des phases de description des UPV et SPV.

2.2.4.1 Processus

Diane+ envisage l'implication des acteurs du développement sans distinguer de rôle particulier (voir tableau 2.10). Par ailleurs, il n'est pas fait mention de collaborations, ni de la démarche que devrait adopter le chef de projet ou l'ingénieur de méthode pour intégrer le fragment Diane+ dans Merise ou toute autre méthodologie.

Tableau 2.10 – Grille d'analyse du processus de la méthode Diane+

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	–	–	–
Spécification des besoins	ND	ND	–
Analyse	–	–	–
Conception	–	–	–

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome ; ND : Non déterminé

2.2.4.2 Modèles

Les langages et modèles utilisés dans la méthode Diane+ ont varié au long de son histoire. Ainsi, bien que l'évolution entre Diane et Diane+ soit marquée par l'intégration du

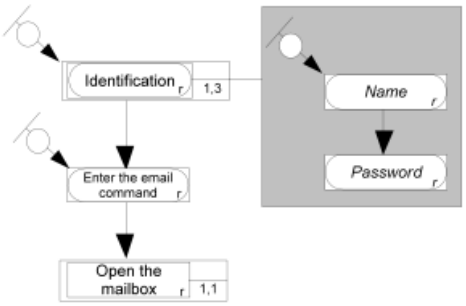


FIGURE 2.6 – Modélisation d'une tâche de consultation de boîte aux lettres électronique avec la notation DIANE

modèle de données OPAC [116], cette dernière notation semble avoir été abandonnée dans les versions les plus récentes de la méthode. Demeure la notation DIANE+, au cœur de la méthode.

La figure 2.6 montre un exemple de mise en œuvre de la notation initialement associée à DIANE. Le diagramme modélise une tâche de consultation d'une boîte aux lettres électronique. Les deux premiers éléments de la partie gauche du diagramme décrivent des tâches interactives et requises. Le dernier élément décrit une réponse du système (ouvrir la boîte). Cette représentation est axée sur la décomposition des tâches utilisateur, la représentation des entrées et les réponses du système. Elle peut donc être comparée aux modèles de tâches tels que CTT.

Deux autres modélisations sont également adoptées par DIANE+ : le prototypage, utilisé afin de valider les spécifications décrites avec la notation DIANE+, auprès des utilisateurs, et les scenarii. Ces derniers sont élaborés en collaboration avec les utilisateurs, comme raffinement des tâches identifiées à l'aide de la notation. Il est également proposé de réaliser des entretiens avec les utilisateurs, comme alternative aux scenarii.

Nous résumons dans le tableau 2.11 les caractéristiques des modèles utilisés dans la version la plus récente de DIANE+ [115].

TABLEAU 2.11 – Grille d'analyse des modèles de la méthode DIANE+

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL classique			
Modèle IHM		Diane+	
Scénario		{Comm}	
Prototype		{Comm}	

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

2.2.4.3 Outils

Les contributions concernant la méthode Diane+ ne font pas explicitement référence à un outillage. En revanche, les travaux concernant l'adaptation des spécifications à des contextes d'interaction concrets variés (interaction sur téléphone portable, Smartphone, station fixe etc.) suggèrent d'automatiser le processus d'implémentation. Toutefois, les auteurs ne semblent pas proposer de contribution allant dans ce sens.

Tableau 2.12 – Grille d'analyse de l'instrumentation de la méthode Diane+

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
–	–	–

2.2.5 Ovid/User Engineering

Ovid/User Engineering est une méthode élaborée au sein d'IBM Corporation, en tant que partie du « Common User Access » [6], dont la première ébauche date du début des années 1990. Elle a intègre la modélisation UML à la fin de cette décade et prend alors le nom d'OVID (Object, View and Interaction Design).

Parmi la rare littérature sur les applications de Ovid/User Engineering, un article de D. Corlett [33] décrit l'application de la méthode sur un système mobile tactile destiné aux jeunes enfants, démontrant ainsi l'applicabilité de la méthode pour le développement d'interfaces non traditionnelles.

Dans son plus récent avatar, et sous le nom de User Engineering, la méthode est constituée de six phases, reprises en figure 2.7. Le détail des activités constituant les trois premières phases est présenté en figure 2.8.

User Engineering définit une phase d'analyse du métier existant, dénomée « Opportunités métier » (c.f. figure 2.8(a)). Elle fournit l'occasion aux spécialistes métier de définir, en concertation avec les décideurs, les objectifs de l'application. De plus, les activités User Engineering et les produits effectivement mis en œuvre dans le développement sont sélectionnés au cours de cette activité.

Les objectifs des décideurs sont structurés sous forme de diagrammes de classes, dont la figure 2.9 présente une illustration. Chaque objectif est défini et une mesure permettant de quantifier la satisfaction de l'objectif lui est associée.

La phase « Compréhension des utilisateurs » (c.f. figure 2.8(b)) consiste à proposer un modèle des utilisateurs. Celui-ci est construit de façon similaire au modèle des objectifs décideurs. Chaque objectif utilisateur est lié à un objectif décideur et raffiné en tâches utilisateurs décrivant le processus actuel permettant de satisfaire l'objectif. Sur cette base, des cas d'utilisation au sens UML sont décrits, pour chaque objectif pour lequel une évolution de l'activité métier est envisagée.

La phase de « Conception initiale » guide la description des éléments de l'interface utilisateur. Les concepts (« objets utilisateur » ou « objets de l'interaction ») effectivement manipulés par

6. Ensemble cohérent de règles, publié par IBM Corporation en 1987 [12], définissant des comportements et rendus graphiques homogènes censés apporter une harmonisation aux applications DOS.

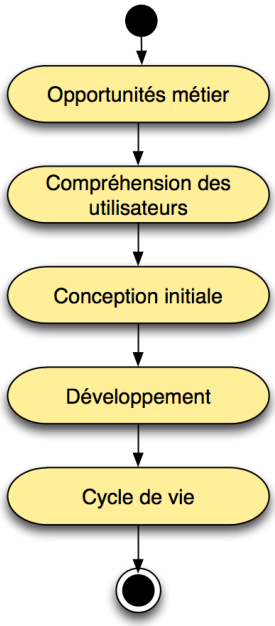


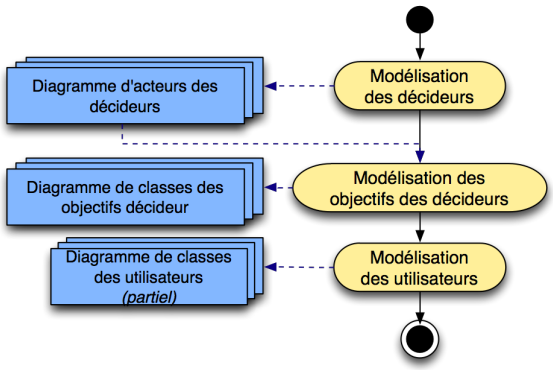
FIGURE 2.7 – Processus User Engineering

les utilisateurs sont d'abord représentés sous forme de diagrammes de classes. Une nouvelle organisation des tâches est ensuite modélisée, sous forme de diagramme d'activités, avec pour objectif de corriger les éventuels défauts des systèmes pré-existants (le cas échéant). Une représentation abstraite de l'IHM est ensuite modélisée sous forme de diagrammes de classes, dans lesquels les propriétés interactionnelles des objets représentés sont décrites (c.f. figure 2.10). Enfin, les objets de l'interaction et les vues sont raffinés, des diagrammes d'états peuvent éventuellement compléter la description des objets utilisateur, puis associés aux technologies et techniques GL adéquates pour implémenter le système. Il est précisé que, lors de l'implémentation, les objets de l'interaction complets peuvent être implémentés comme des objets métier au sein du projet.

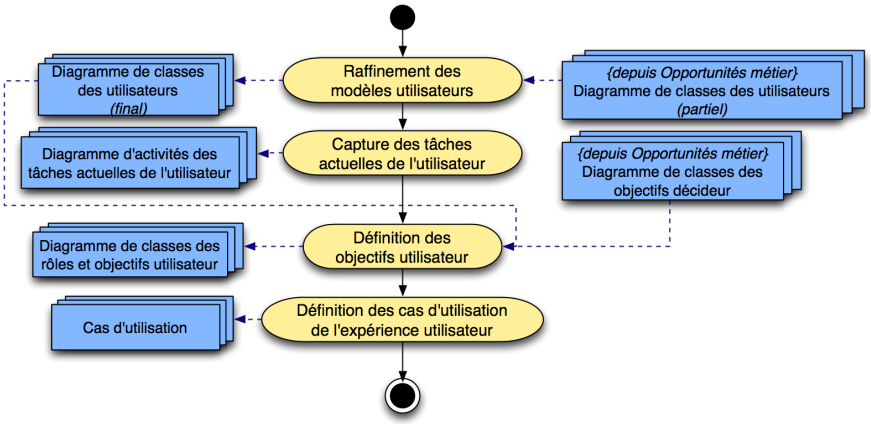
Les mécanismes d'itération de la méthode sont peu abordés. Tout au plus est-il fait mention de la possibilité de reprendre l'amont du cycle de développement lorsque de « nouvelles découvertes » sont faîtes quant au projet. Le passage d'une phase à l'autre est verrouillé par une validation des produits, ou tout du moins une revue globale des produits. La validation peut notamment donner lieu à la réalisation d'un prototype dit de « basse fidélité », afin de faciliter la communication des solutions de conception auprès des décideurs et utilisateurs.

Les caractéristiques de User Engineering sont les suivantes :

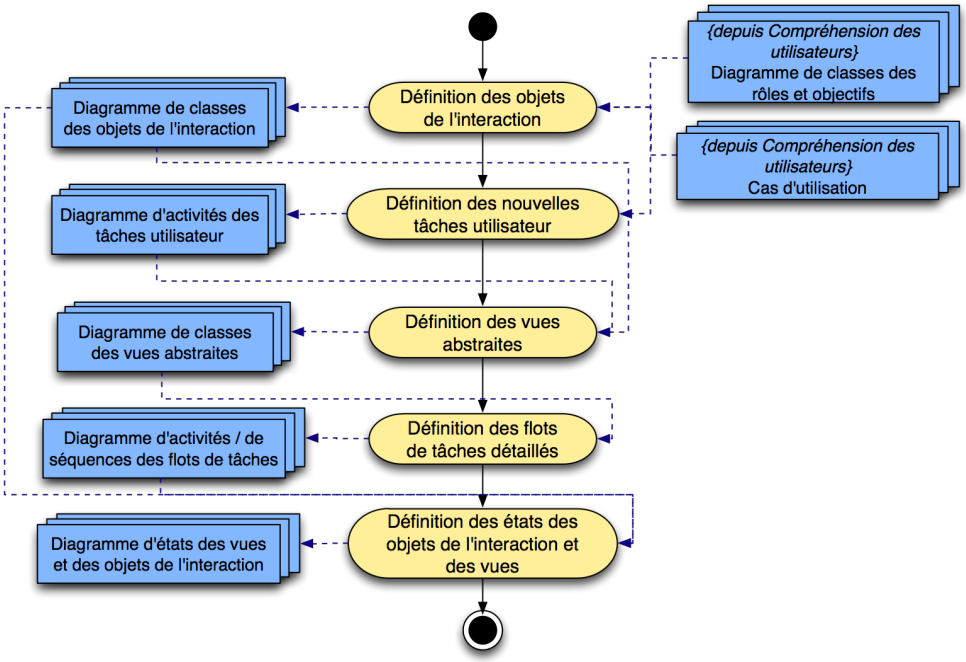
- les entités de premier ordre sont les *objectifs et valeurs* associées aux décideurs, décrits dans le « diagramme de classes des objectifs décideurs » ;
- la méthode est *centrée* sur l'*utilisateur et les décideurs* ;
- le cycle de développement est *itératif*, globalement *parallèle* et *séquentiel* au niveau local (notons que les activités au sein d'une même phase puissent être développées en parallèle).



(a) Phase d'opportunités métier



(b) Phase de compréhension des utilisateurs



(c) Phase de conception initiale

Figure 2.8 – Détail des phases de la méthode User Engineering

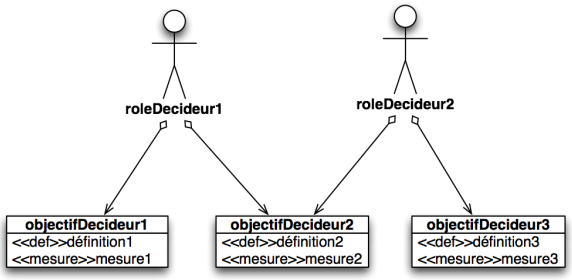


FIGURE 2.9 – Exemple d'objectifs décideurs, d'après D. Roberts [97]

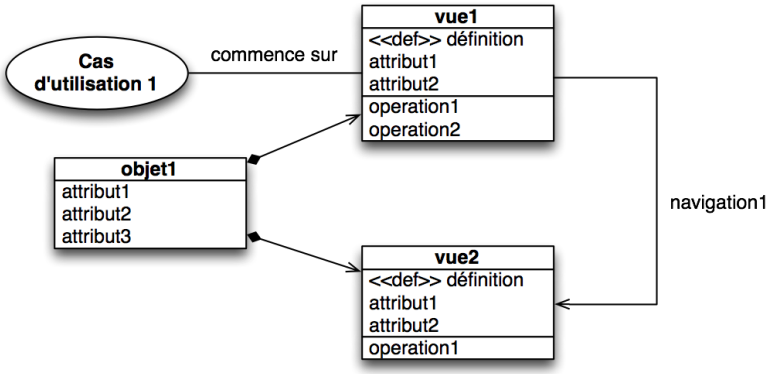


FIGURE 2.10 – Fragment de diagramme objet-vue, d'après D. Roberts [97]

2.2.5.1 Processus

Dans sa configuration actuelle, User Engineering implique 6 rôles dans le développement logiciel. Nous présentons dans le tableau 2.13 une estimation de l'implication des acteurs au cours du cycle de développement. En effet, nous n'avons identifié aucune description claire des rôles fonctionnels auxquels se rattachent les activités User Engineering. Nous les avons donc déduits à partir des responsabilités spécifiques décrites par D. Roberts [97].

Du fait que l'ensemble des phases de User Engineering donne lieu à des modélisations UML d'une part, et du fait que les responsabilités liées aux prescriptions ergonomiques semblent relativement simples d'autre part (il n'est pas fait mention des pratiques d'évaluation ergonomique), nous estimons que le développement des aspects IHM du système est affecté à des acteurs correspondant au rôle fonctionnel du spécialiste IHM.

2.2.5.2 Modèles

User Engineering « s'appuie sur un sous-ensemble d'UML afin de fournir un langage décrivant la conception de façon non ambiguë » [7] [97]. Les modèles utilisés s'appuient pour l'essentiel sur les diagrammes de classes, diagrammes d'états, diagrammes d'activités et diagrammes de séquences.

7. "[User Engineering] relies on a subset of UML to provide a langage to portray the design in an unambiguous way."

Tableau 2.13 – Grille d'analyse du processus de la méthode Ovid/User Engineering

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	M	–	–
Spécification des besoins	I	I	–
Analyse	I | G	I	–
Conception	G	G	–

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome

L'ensemble des modèles élaborés au cours du développement peuvent être utilisés comme des modèles communicationnels. En effet, la relative simplicité du sous-ensemble d'UML adopté est considérée comme adéquate pour permettre les échanges avec utilisateurs et décideurs. De plus, l'équipe de conception de l'IHM est considérée comme « guardienne » ("gatekeeper") des connaisssances accumulées dans les modèles construits, auprès des utilisateurs et des décideurs. Lorsque la complexité du système compromet cette communication, il est préconisé d'avoir recours à toute forme de communication adaptée, notamment le prototypage. Toutefois, pour autant que le sache l'auteur, aucune indication claire n'est donnée quant à la mise en œuvre de ces médias.

Le tableau 2.14 résume les considérations exprimées ci-dessus.

Tableau 2.14 – Grille d'analyse des modèles de la méthode Ovid/User Engineering

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL classique	UML{Comm}	UML{Comm}	UML (mise en relation des modèles GL et IHM)
Modèle IHM			
Scénario	(Cas d'utilisation){Comm}		
Prototype	Comm	Comm	

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

2.2.5.3 Outils

Le recours aux outils est central dans Ovid/User Engineering. Roberts recommande leur utilisation systématique, à toutes les étapes du développement, aussi bien pour faciliter la gestion du planning, la communication que pour concevoir le système lui-même (i.e., via des outils d'aide à la conception).

D'autre part, la démarche d'utilisation des outils est également décrite : il est ainsi recommandé de construire les modèles de niveau implémentation à l'aide du même outil que pour les modèles de niveau spécification.

TABLEAU 2.15 – Grille d'analyse de l'instrumentation de la méthode User Engineering

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
–	Reprise des outils RUP	–

2.2.6 Wisdom

La méthode Wisdom, proposée par N. J. Nunes, intègre la conception centrée usage et le développement orienté objet. Elle s'adresse en particulier aux petites équipes de développement dont l'objectif est de *« concevoir, construire et maintenir des systèmes interactifs avec les plus hauts standards de qualité produit et processus »* [8][83].

Afin d'atteindre ces niveaux de qualité, la méthode est construite autour des principes du *prototypage rapide* et du *développement centré utilisateur*. Ce dernier critère est obtenu en intégrant :

- les travaux de Constantine[29] (c.f. chapitre suivant) sur le développement des cas d'utilisation sous forme de *« cas d'utilisation essentiels »* [9],
- le modèle de présentation issue de la méthode Ovid/User Engineering[97] (voir section 2.2.5.2),
- la terminologie de modélisation du Unified Process [62],
- le cycle de développement – plus particulièrement, le fait de baser les itérations sur un raffinement de prototypes convergeant vers le produit final – ainsi que la description des cas d'utilisation sous forme de « flots de tâches » de la méthode Bridge [41],

Wisdom s'inscrit implicitement dans l'héritage des méthodes « légères » (c'est-à-dire, peu prescriptives, ayant recourt à un nombre réduite voire inexistant de produits), les plus représentatives aujourd'hui étant les processus se réclamant du Manifeste Agile, telles que Crystal Clear [27], Scrum [100], ou eXtreme Programming [9]. Nunes précise la notion de « méthode légère » en indiquant que ce type de méthode peut être apprise puis appliquée en quelques jours, par opposition aux processus plus lourds, nécessitant une formation longue et entraînant une mise en place complexe. Ainsi, l'auteur suggère que, partant d'une absence d'organisation claire et d'un développement à l'organisation aléatoire [10], les équipes de développement adoptant Wisdom soient formées au fur et à mesure des besoins de formalisation du processus de développement.

L'un des éléments clef de la méthode réside dans son approche itérative. En effet, la dynamique de Wisdom s'appuie largement sur le prototypage dit « évolutionnel » : si tous les types de prototypage sont encouragés (prototypes exploratoires, prototypes expérimentaux), une emphase particulière est apportée sur les prototypes dits « systèmes pilotes », c'est-à-dire *« des prototypes très matures pouvant être déployés pour un usage effectif »* [11] [83].

8. *"[Wisdom] address[es] the specific needs of small development teams who are required to design, build and maintain interactive system[s] with the highest process and product quality standards"*
9. *"essential use cases"*
10. *"random acts of hacking" [83]*
11. *"Pilot systems [..] are very mature prototypes that can actually be deployed for effective use"*

Le cycle de développement de Wisdom remplace la traditionnelle phase d'implémentation par une discipline d'« évolution en rapides » (*whitewater evolution*). Cette notion est issue de la méthode Bridge [41] ; son appellation souligne l'analogie entre *« un processus de développement comportant de fortes ressemblances avec l'énergie intense des torrents où l'ensemble de l'énergie (apparemment désordonnée) permet à l'activité dans son ensemble de progresser très rapidement »* [12] [83]

Un des points essentiels différenciant Wisdom de méthodologie telles que RUP est la collecte des besoins utilisateurs. En effet, là où ces dernières prône la description de cas d'utilisation, Wisdom suggère la description de « flots de tâches utilisateur essentielles ». Celles-ci synthétisent l'approche par « cas d'utilisation essentiels » de Constantine et la description des tâches prônée par la méthode Bridge. De la première, Wisdom adopte la sélection des besoins utilisateurs apportant une valeur indéniable à ces derniers. De la seconde, la méthode intègre la description des cas d'utilisation sous la forme de diagrammes d'activités de haut niveau, divisés en actions utilisateur et responsabilités système. La figure 2.11 présente un exemple de flot de tâches utilisateur essentielles.

La description des flots de tâches utilisateur essentielles, réalisée au cours du processus de collecte des besoins (c.f. figure 2.12(a)), donne lieu à la construction de prototypes abstraits utilisés à des fins de validation auprès des utilisateurs (selon l'approche préconisée par Constantine et al. [29]).

Lors du processus d'analyse (c.f figure 2.12(b)), les concepteurs s'appuient sur la partie système du flot de tâches pour définir l'organisation des classes d'analyse. La partie interaction du flot de tâches détermine le modèle d'interaction en terme de tâches et d'espaces d'interaction. Une fois l'architecture interne et l'architecture d'interaction construites, celles-ci sont reliées par des associations de communication ou de souscription. L'objectif de cette activité est de satisfaire les besoins exprimés dans les cas d'utilisation essentiels développés au cours de l'itération actuelle.

Enfin, la phase de conception (c.f. figure 2.12(c)) préconise le raffinement des modèles d'analyse et d'interaction, pour le premier en intégrant l'architecture technique (langages de programmation, outils, technologies de persistance etc.), le second en décrivant un modèle de présentation (c'est-à-dire, l'organisation des espaces de travail) ainsi qu'un modèle de tâches (c'est-à-dire, l'organisation des tâches utilisateur).

L'organisation des processus est décrite en figure 2.13, possède les caractéristiques suivantes :

- Les entités de premier ordre sont les *flots de tâches utilisateur essentielles* (comparables aux « cas d'utilisation essentiels » de Constantine [29]),
- La méthode est *centrée* sur l'*usage*,
- Le cycle de développement est *itératif*, globalement *incrémental* et localement *parallèle*.

2.2.6.1 Processus

L'ensemble des acteurs du développement de Wisdom semble être limité au rôle fonctionnel du spécialiste GL. Il n'est effectivement pas fait mention d'ergonome ou de spécialiste IHM. Ce phénomène nous semble cohérent avec le premier principe de Wisdom, à savoir

12. *"a development process with high resemblance to the intense energy in rapid streams of water where the total (and apparently messy) energy makes the activity as a whole progress very quickly"*

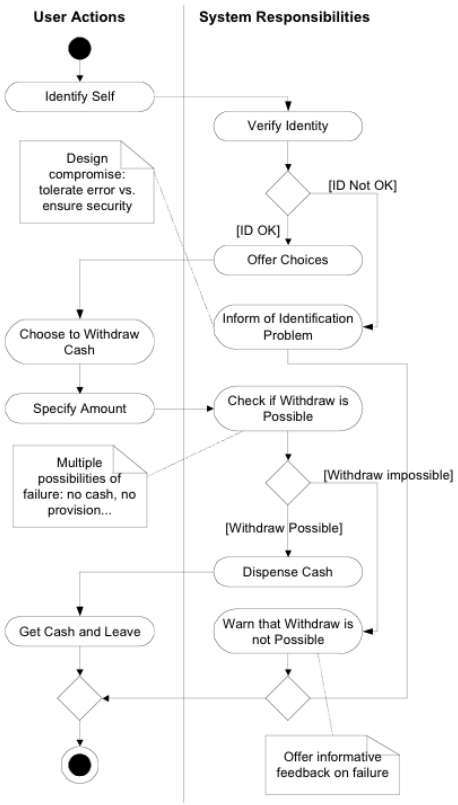


FIGURE 2.11 – Flot de tâches utilisateur essentielles Wisdom : exemple de système de retrait d'argent, d'après Nunes [83]

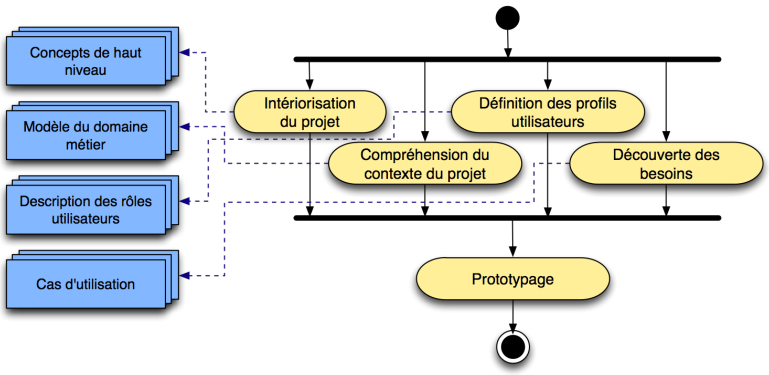
que la méthode est destinée à de petites équipes de développement très dynamiques. Subséquemment, les rôles des différents acteurs ne sont pas clairement distingués au cours du développement, bien que l'auteur mentionne l'expertise de l'ergonome, du concepteur d'interfaces homme-machine et du spécialiste du génie logiciel.

Le développement de l'IHM est clairement identifié, dans les phases d'analyse et de conception (c.f. figures 2.12(b) et 2.12(c)), en parallèle au développement des fonctionnalités internes du système.

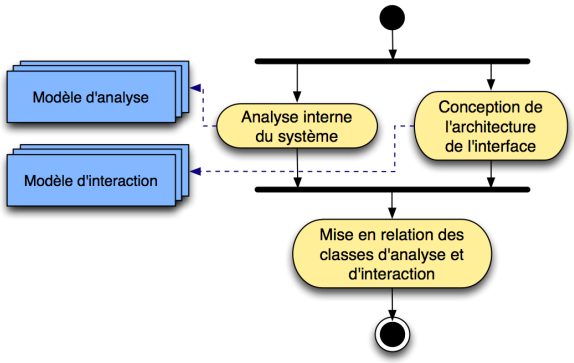
Le tableau 2.16 décrit ainsi un processus reposant exclusivement sur le spécialiste GL, au sein duquel aucune collaboration ne semble se dérouler. Dans les faits, l'auteur prône le développement logiciel dans des locaux non cloisonnés (open spaces), dans lesquels les échanges entre acteurs du développement sont nombreuses et systématiques. Toutefois, le but de ces échanges n'est pas clairement identifié. Par conséquent, les collaborations, si elles ont lieu, se focalisent sur les activités de construction et de mise en relation de modèles, sans pour autant établir les modalités de ces collaborations (voir section 2.2.6.2).

2.2.6.2 Modèles

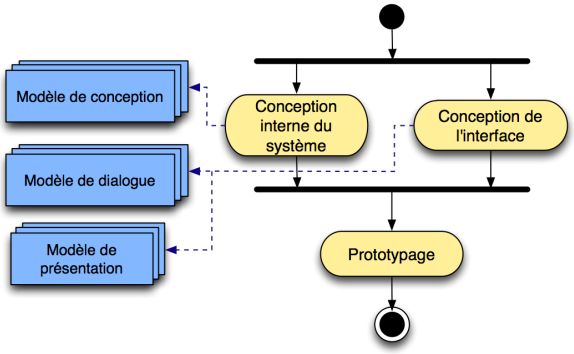
Les modèles décrits au cours de l'application de la méthode Wisdom se rattachent tous à la notation UML, dont certains diagrammes sont éliminés ou réorganisés afin de fonder la « notation Wisdom ». Par conséquent, cette notation constitue pour l'essentiel une



(a) Processus de collecte des besoins



(b) Processus d'analyse



(c) Processus de conception

Figure 2.12 – Phases de la méthode Wisdom

simplification d'UML, dans le respect de la philosophie minimaliste à la base de Wisdom. En revanche, le remplacement des cas d'utilisation UML par une représentation des flots de tâches utilisateur « essentielles » (voir figure 2.11) constitue une différence notable et particulière de l'approche Wisdom. Ce dernier modèle est notamment utilisé à des fins de validation auprès des utilisateurs finaux.

Quant à la représentation de l'IHM, Wisdom s'appuie sur trois diagrammes : une représentation des tâches utilisateur, un modèle du dialogue et un « modèle de présentation », ces deux-ci étant déduits du premier.

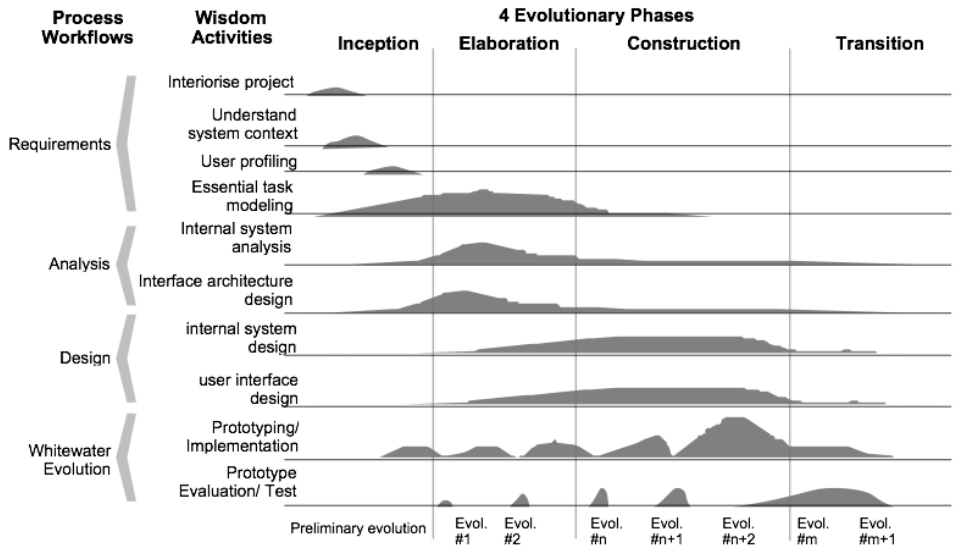


FIGURE 2.13 – Organisation de la méthode Wisdom, tiré de Nunes [83]

TABLEAU 2.16 – Grille d'analyse du processus de la méthode Wisdom

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	G	–	–
Spécification des besoins	G	G	–
Analyse	G	G	–
Conception	G	G	–

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome

Le premier diagramme est en fait une traduction de la représentation du formalisme CTT (c.f. chapitre suivant) sous la forme de diagrammes de classes UML stéréotypés. Cette approche permet notamment aux spécialistes GL impliqués dans le développement de manipuler les concepts CTT sans devoir auparavant en apprendre le formalisme.

Le modèle de présentation correspond à un découpage des espaces de travail déduit de la structure du modèle de tâches, selon la même approche que celle préconisée par la méthode de conception des IHM issue du projet CAMELEON [4] (c.f. chapitre suivant).

De même, le modèle du dialogue correspond à une vue de la dynamique de l'interface homme-machine, également déduite du modèle des tâches.

2.2.6.3 Outils

Grâce à l'héritage de UP et UML, Wisdom peut être instrumenté avec les mêmes outils que ceux utilisés pour la méthode RUP, par exemple.

Tableau 2.17 – Grille d'analyse des modèles Wisdom

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL	UML{Comm}	UML	UML(Associations entre diagrammes de classes GL et IHM)
Modèle IHM			
Scénario			
Prototype	Validation générale {Comm}	Validation des solutions d'interaction {Comm}	

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

De plus, l'intégration de la notation CTT au sein d'UML évite aux concepteurs d'avoir recours à un outillage tiers.

Enfin, Nunes souligne la nécessité de proposer, au-delà de l'outillage, une infrastructure de modélisation commune à l'ensemble des outils manipulant UML. Selon l'auteur de la méthode, cette infrastructure, basée sur le format d'échange XMI [88], est nécessaire dans les scenarii d'usage suivants : la combinaison d'outils dans des environnements de développement hétérogènes, la coopération dirigée par des métamodèles communs, le travail dans des environnements asynchrones et la promotion des patrons de conception et de la réutilisation. Cette recommandation semble constituer l'une des premières mentions d'intégration technique de modèles au sein d'un processus de conception.

Tableau 2.18 – Grille d'analyse de l'instrumentation de la méthode Wisdom

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
–	Utilisation de l'outillage UML standard	–

2.2.7 UCD Agile

UCD Agile (*User Centered Development Agile*) décrit moins une méthode qu'un regroupement d'expériences sur l'intégration d'un ergonome et de la conception centrée utilisateur dans un cycle de conception Agile [Agi].

Les méthodes agiles sont caractérisées par :

- des cycles de développement itératifs très courts : chaque projet est découpé en incréments délivrables en un intervalle de temps allant de quelques jours à quelques semaines ;
- les cycles de développement ne comportent pas de phase à proprement parler : une fois les incréments du projets définis, ceux-ci sont priorisés par l'équipe de développement et

les décideurs, puis des tests de validation sont décrits pour chaque fonctionnalité, puis ces dernières sont implémentées ;
- les incréments du développement correspondent aux « histoires d'utilisateurs » (*user stories*) : des fragments de fonctionnalités, résumés sur des fiches individuelles, servant de base aux échanges entre utilisateurs et équipe de développement ;
- des réunions régulières (quotidiennes ou tous les deux jours) de l'équipe de développement, afin d'établir l'avancement de tous les incréments en cours de développement, les prévisions de travail de chaque participant, ainsi que les problèmes rencontrés ;
- une implication des utilisateurs (clients, décideurs et utilisateurs finaux) comme partie intégrante de l'équipe de développement : ces derniers, par l'intermédiaire d'un représentant, sont des interlocuteurs privilégiés de l'équipe de développement, impliqués à la fois pour la définition du périmètre du projet, comme personnes ressources lors de la conception et en tant que qu'instances validantes.

Considérant ces caractéristiques, les méthodes agiles apparaissent comme partageant des points communs avec la conception centrée utilisateur, en particulier la place prise par l'utilisateur dans la conception, ainsi que l'aspect itératif des projets de conception.

Les divergences entre les deux approches proviennent d'abord de la construction des démarches : la conception centrée utilisateur intègre une activité, souvent longue, d'études ethnographiques et de profilage de l'utilisateur ; les méthodes agiles, en revanche, laissent aux utilisateurs le soin d'exprimer leurs attentes, et éliminent ainsi une large partie des analyses conceptuelles menées dans les méthodes « classiques ». De plus, la durée des itérations diffère entre les deux approches : contrairement aux méthodes agiles, la conception centrée utilisateur intègre dans chaque itération des activités d'enquête contextuelle, d'élaboration des prototypes, puis d'évaluation utilisateur, couvrant plusieurs semaines.

Les travaux de F. Maurer, décrits dans Fox et al. [51] rassemblent et analysent des expériences d'intégration d'un rôle « ergonome », mettant en œuvre les principes de la conception centréee utilisateur dans les équipes de développement agile. Le cadre de conception proposé par les auteurs et illustré en figure 2.14 suggère l'instauration d'une « étape initiale » (*initial stage*) de conception, au cours de laquelle l'ergonome réalise enquêtes contextuelles, conception de l'IHM et réalisation d'un prototype basse fidélité (c'est-à-dire, une maquette crayonnée de l'interface), lequel est transféré aux développeurs.

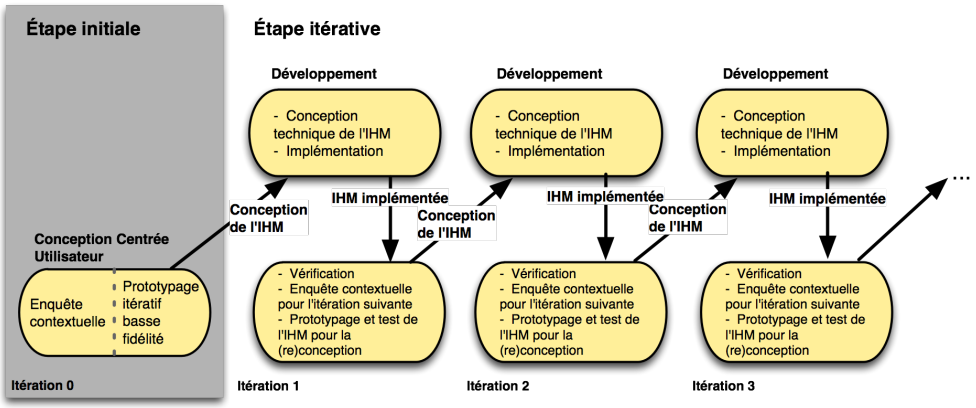


FIGURE 2.14 – Cycle de développement d'un projet UCD Agile, d'après Fox et al. [51]

Fox et al. [51] relèvent une durée moyenne de réalisation de l'étape initiale de quatre se-

maines, très inférieur à celles observées dans les approches de conception centrée utilisateur traditionnelles. Au cours de ce temps réduit, l'ergonome ne réalise pas un prototypage de l'intégralité du système, mais propose un premier ensemble d'artefacts qui sera complété au cours de l'étape suivante.

À la suite de cette première activité, le processus UCD Agile entre dans l'« étape itérative » (*iterative stage*). Cette phase du développement comprend l'implémentation du concept d'IHM proposé par l'ergonome, ajouté aux étapes classiques de conception agile (capture des besoins utilisateurs, description de tests de validation, implémentation). En parallèle à l'implémentation d'un incrément, l'ergonome évalue les délivrables existants et approfondit les enquêtes contextuelles, en anticipation des itérations futures.

Contrairement à la linéarité des enchaînements d'activités suggérée par la figure 2.14, les processus de conception centrée utilisateur et de conception agile ne sont que rarement synchronisés : les contributions de l'ergonome peuvent être délivrées en cours d'itération et implémenté à la suivante, et les résultats des évaluations peuvent être produits avec plusieurs itérations de décalage.

Sur l'ensemble des expériences recueillies par Fox et al., l'investissement de plusieurs semaines consacré à l'étape initiale est considéré comme valable, c'est-à-dire qu'il permet de cibler plus nettement les attentes des utilisateurs, et par conséquent de délivrer une plus grande valeur ajoutée pour chaque incrément du projet.

Nous résumons ci-dessous les caractéristiques de la méthode :

- les entités de premier ordre sont les *histoires d'utilisateurs* (*user stories*) ;
- la méthode est centrée sur l'*utilisateur* ;
- le cycle de développement est *itératif* et *incrémental* ; les activités de développement sont réalisées en *parallèle*.

2.2.7.1 Processus

Les caractéristiques du processus UCD Agile sont décrites dans le tableau 2.19. Nous considérons que les activités d'enquête contextuelle et d'élaboration de concepts d'IHM, menées par l'ergonome, consistent en des activités de modélisation métier et de spécification des besoins. Les développeurs Agile, considérés comme des spécialistes GL, réalisent également une spécification des besoins, sous la forme d'histoires d'utilisateurs. Ces dernières sont rédigées en collaboration avec les utilisateurs et les ergonomes ; nous signalons donc une collaboration lors de la spécification des besoins.

2.2.7.2 Modèles

Les méthodes agiles n'utilisent pas de modélisation GL à proprement parler. Selon cette philosophie, UCD Agile n'utilise pas non plus de modèle de l'IHM. Les seuls modèles utilisés (c.f tableau 2.20) sont les histoires d'utilisateurs (assimilables à des scenarii) et les prototypes basse fidélité (maquettes crayonnées de l'interface).

TABLEAU 2.19 – Grille d'analyse des processus UCD Agile

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	E	E	
Spécification des besoins	G | E	E	G,E
Analyse	G	G	
Conception	G	G	

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome

TABLEAU 2.20 – Grille d'analyse des modèles UCD Agile

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL			
Modèle IHM			
Scénario	{Comm}		
Prototype	{Comm}		

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

2.2.7.3 Outils

En support des cycles de conception UCD Agile, F. Maurer propose l'outil ActiveStory Enhanced [59]. Il s'agit d'un outil pour le maquettage, permettant de réaliser des prototypes basse définition navigables, ayant l'apparence de crayonnés. Cette approche prévient le biais introduit par les prototypes trop réalistes, qui ont une certaine tendance à inciter les utilisateurs à se préoccuper précocement de détails graphiques non pertinents (couleurs, forme des interacteurs etc.).

ActiveStory Enhanced intègre également des outils de collectes de métriques, telles que la durée de consultation de chaque écran, les clics souris, les traces d'exécution etc.

TABLEAU 2.21 – Grille d'analyse de l'instrumentation UCD Agile

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
	ActiveStory Enhanced	

2.3 Analyse critique des pratiques d'ingénierie des méthodes GL/IHM

L'analyse des mécanismes sous-tendant les différentes méthodologies présentées précédemment nous permet d'identifier plusieurs mécanismes organisationnels récurrents. Leur regroupement constitue ainsi une compilation de pratiques, bonnes ou discutables, d'ingénierie des méthodes.

2.3.1 Généralités

Au cours de la quinzaine d'années d'évolution des méthodes couverte par notre analyse, la souplesse des démarches de développement ainsi que l'implication des utilisateurs dans le processus sont progressivement devenues des préoccupations prédominantes.

Il est tentant d'associer ce phénomène à la publication du rapport « CHAOS » du Standish Group, en 1995 [111]. Celui-ci fait état d'un taux d'échec des projets de développement informatique alarmant : 31,1% des projets sont annulés et 52,6% font face à des dépassements de coût (189% en moyenne), d'agenda de livraison (222% en moyenne) ou présentent des fonctionnalités incomplètes (61% des fonctionnalités annoncées présentes dans le produit final).

Le rapport conclut son analyse en identifiant trois critères essentiels de réussite pour les projets informatiques : 1) l'implication des utilisateurs dans le processus de développement; 2) le soutien de la part du management exécutif des organisations commanditaires et 3) la définition claire des besoins. [13]

Parmi les trois points susmentionnés, la définition claire des besoins (et l'adaptation du cycle de développement à leur évolution), et l'implication des utilisateurs dans le processus de développement sont directement liés à l'ingénierie des méthodes.

2.3.1.1 Souplesse des méthodologies

En quinze années, les méthodes de développement ont évolué depuis l'encadrement des paradigmes procéduraux vers les paradigmes objet. Alors peu usitées pour le développement des systèmes basés sur le paradigme procédural (C, Cobol, Fortan…), les méthodes de développement ont été plus largement employées sur les systèmes basés sur le paradigme objet (C++, Java, .Net…) [111].

D'autre part, des travaux tels que les études du Standish Group ont permis d'identifier les causes d'échec des projets de développement informatique, traçant par là-même la silhouette de nouvelles méthodes de développement, intégrant le caractère intrinsèquement versatile des besoins utilisateur et des spécifications.

Concrètement, l'étude des pratiques de développement ont révélé que les méthodes trop formelles et/ou trop figées (c'est-à-dire, peu adaptées au développement itératif ou incrémental) sont peu utilisées. À cause du manque de documentation concernant la mise en

13. Au cours des rapports qui ont suivi [112, 113], dont les chiffres marquent une amélioration progressive mais indéniable des chiffres présentés, le troisième point a été remplacé par 3) la présence d'un chef de projet expérimenté, soulignant ainsi la prééminence du facteur humain dans les processus de développement.

œuvre de la méthode, alors apparemment focalisée sur le développement de projets de taille importante, RUP a ainsi souffert d'une réputation de méthode « monolithique ».

Pourtant, la méthode propose des plans d'itération permettant d'adapter le développement à la fois aux caractéristiques du projet et aux blocs fonctionnels en cours de développement. La même approche est adoptée par les méthodes User Engineering et Wisdom, qui préconisent de limiter les activités et les produits au minimum nécessaire pour permettre un développement effectif et efficient. Ces démarches sont alors adaptées en fonction de la complexité du projet, de la maîtrise du domaine métier par l'équipe de développement ou encore l'expérience de cette dernière.

Les méthodes UPi, DIANE+, User Engineering, Wisdom et UCD Agile proposent toutes quatre un cycle de développement itératif, permettant de raffiner l'application en fonction des besoins émergents des utilisateurs. Hormis User Engineering, les méthodes étudiées décrivent les applications de manière incrémentale, afin de fournir au plus tôt des blocs applicatifs opérationnels. Quant à l'effort de développement au niveau activités, aucun consensus ne semble émerger clairement : UPi et User Engineering stipulent que le passage d'une phase du développement à l'autre est conditionné par la validation des produits réalisés durant la phase. En revanche, l'enchaînement des activités au sein d'une phase ne semble pas être contraint : leur réalisation intervient dès que possible. Cette approche se retrouve dans les méthodes RUP et Wisdom, ainsi que dans les processus agiles (tel UCD Agile), où les activités de spécification sont déclenchées dès l'expression du premier besoin.

À l'inverse des adaptations dirigées par les processus, Wisdom et UCD Agile focalisent le développement et l'itération sur les prototypes. Les rôles fonctionnels du développement de ces méthodes sont peu formalisés – hormis le rôle d'ergonome dans UCD Agile–, et l'ensemble du processus est adapté aux besoins organisationnels de l'équipe de développement, dont les capacités d'adaptation déterminent l'issue de la conception et de l'implémentation du système.

Suivant cette optique, la méthode déroule peu d'étapes de raffinement (description des tâches essentielles et des acteurs, analyse puis implémentation) et s'appuie sur un nombre restreint de modélisations (4 types de diagrammes UML déclinés en 8 modélisations

User Engineering adopte la même approche, en affirmant que les équipes de développement ne doivent sélectionner que les activités dont elles ont besoin. Le retour d'expérience décrit pas Corlett [33] confirme qu'une application trop dogmatique de la méthode nuirait à la créativité nécessaire pour envisager des interfaces homme-machine innovantes.

Pour résumer

Les processus de développement proposent une plus grande souplesse méthodologique qu'auparavant. Les démarches de développement s'adaptent aux différents facteurs caractérisant tout projet informatique : la complexité du domaine métier, l'expertise de l'équipe de développement, sa taille etc.
De plus, elles sont caractérisées par un cycle de développement itératif, incrémental, et maximisant le parallèlisme des activités, marquant ainsi une tendance à fournir aux utilisateurs et décideurs matière à valider (et, le cas échéant, matière à itérer pour l'équipe de développement), afin de délivrer rapidement des blocs applicatifs.

2.3.1.2 Implication des utilisateurs dans le développement

L'implication des utilisateurs dans le développement logiciel est une problématique de recherche traitée depuis de nombreuses années par le domaine de l'ingénierie des besoins [85]. Dans le contexte des méthodes d'analyse GL/IHM, nous distinguons deux approches : d'une part la spécification des besoins en fonction de la valeur attendue par l'utilisateur, d'autre part l'implication personnelle et physique des utilisateurs dans le quotidien du développement. La première approche est fortement empreinte des travaux de Constantine [29] sur les « cas d'utilisation essentiels », et est adoptée par Wisdom, Diane+, UPi et Ovid/User Engineering, que les méthodes agiles (et donc UCD Agile) cumulent avec la seconde.

Les deux types d'approches convergent, en revanche, en ce qui concerne l'évaluation régulière par les utilisateurs de l'évolution du système. Toutes les méthodes proposent ainsi d'utiliser des prototypes fonctionnels du système en cours de développement.

Néanmoins, l'ensemble des méthodes étudiées, toutes relativement récentes, marque une évolution dans la structuration des phases amont du développement, par rapport aux méthodes plus anciennes telles que Merise ou RUP. La tendance qu'affichent ces méthodologies consiste à focaliser les préoccupations des phases de spécification sur la valorisation apportée à l'utilisateur et sur la réalisation de maquettes permettant à l'utilisateur de valider les premières ébauches du système, en rupture d'avec la pratique antérieure d'étude directe des fonctionnalités du système.

Par ailleurs, nous avons vu que RUP ne peut pas être considéré comme un processus centré utilisateur : l'interprétation adoptée quant aux cas d'utilisation considère en effet davantage le système que l'utilisateur. Lemieux et Desmarais [74] ont confirmé ce diagnostic en ajoutant toutefois qu'il est possible d'aménager la démarche pour y intégrer les principes de la conception centrée utilisateur telle que décrite par l'ISO 13407. Ce type de conception rejoindrait alors la première approche susmentionnée.

Comme nous le verrons par la suite, l'implication des utilisateurs est fortement liée au type d'artefact utilisé pour valider l'expression, la spécification et l'implémentation des besoins utilisateur. En effet, nous constatons que les interventions des utilisateurs (et/ou décideurs) dans une optique de validation se déroulent généralement :

- à l'issue de la capture des besoins ;
- dès la spécification d'un bloc fonctionnel du système répondant à un besoin ;
- dès l'implémentation dudit bloc fonctionnel.

POUR RÉSUMER

Il est indispensable d'impliquer les utilisateurs tout au long du processus de développement. Les spécifications du futur système doivent être abordées du point de vue de la valeur perçue attendue par l'utilisateur, tel que le décrit Constantine [29].
D'autre part, il est également nécessaire de proposer régulièrement à l'utilisateur des modèles du futur système permettant la validation.

2.3.2 Analyse des pratiques liées aux processus GL/IHM

Du point de vue des phases couvertes par les méthodes étudiées, la plupart des méthodes abordent, suivant une phase d'assimilation des pratiques existantes du métier et une identification des profils d'utilisateur, la conception par la spécification des besoins utilisateur.

Quant aux rôles fonctionnels impliqués dans le développement, nous avons à nouveau constaté deux types d'approche :

1. Les spécialistes GL sont impliqués dans la conception de l'IHM, notamment en adaptant les modèles de l'interaction homme-machine aux formalismes du Génie Logiciel (typiquement les arbres de tâches traduits du formalisme CTT en UML). La méthode Wisdom adopte cette approche.
2. Les ergonomes et/ou spécialistes IHM réalisent les activités de conception de l'IHM. UPi, DIANE+, User Engineering, RUP et UCD Agile s'inscrivent dans cette tendance.

Ce dernier point induit une problématique d'intégration des artefacts IHM avec les produits de la spécification et de l'analyse du domaine métier. DIANE+ ne décrit pas de démarche pour intégrer les modèles IHM dans la méthode sur laquelle elle se rattache. Au contraire, UPi décrit l'intégration par les patrons d'utilisabilité, lors de l'analyse et conception, dont l'implémentation fonctionnelle constitue un pont vers l'architecture interne de l'application. Wisdom décrit clairement l'intégration des domaines métier et IHM grâce à une mise en relation de leurs modèles de niveau analyse (c'est-à-dire, l'utilisation d'un modèle pivot). Cette mise en relation est facilitée par le fait que les acteurs la réalisant sont issus du domaine du Génie Logiciel. User Engineering propose une mise en relation des vues du système avec les « objets de l'interaction », concepts initialement manipulés par l'utilisateur. lors de la conception initiale – c'est-à-dire, de l'analyse –, et *in fine* décrits comme des objets métier, lors de l'implémentation.

À l'inverse de l'intégration basée sur les modélisations, les méthodes RUP et UCD Agile s'appuient sur les prototypes pour la collaboration entre les domaines fonctionnel et IHM : pour ces deux méthodes, les développeurs sont chargés soit d'intégrer des prototypes logiciels (RUP), soit d'implémenter des concepts d'IHM décrits sous forme de maquettes papier (UCD Agile). Dans les deux cas, l'implémentation réalisée est ensuite évaluée par les utilisateurs.

Nonobstant ces efforts de mise en relation, aucune méthodologie ne décrit clairement les modalités des collaborations donnant lieu aux modèles de mise en relation des concepts métier et IHM.

Pour résumer

Plusieurs méthodes GL/IHM impliquent des spécialistes issus des domaines GL et IHM. Leurs contributions respectives sont généralement réalisées en parallèle, puis mises en commun au cours de la phase correspondant à l'analyse des besoins utilisateur. Toutefois, aucune méthode ne décrit les modalités de cette intégration.

2.3.3 Analyse des pratiques de modélisation

À la suite de l'analyse des processus, nous considérons l'analyse des pratiques de modélisation. Plus particulièrement, nous étudions dans cet intitulé les approches de représentation des besoins utilisateur, le rôle des scenarii et prototypes, et enfin les langages de modélisation des domaines GL et IHM.

2.3.3.1 Représentation des besoins utilisateur

Si toutes les méthodes préconisent de réaliser un découpage conceptuel du système développé, nous n'identifions pas de consensus net sur le formalisme à adopter : RUP s'appuie sur les cas d'utilisation UML traditionnels, critiqués depuis comme une approche centrée sur les fonctionnalités au détriment de la valeur apportée à l'utilisateur.

Ovid /User Engineering et Diane+ évitent cet écueil et représentent le système en amont comme une décomposition de buts et de tâches utilisateur et/ou décideurs, indépendamment des fonctionnalités du système. À la suite de cette organisation, User Engineering préconise de traduire les buts et tâches en cas d'utilisation. De même, UPi, propose d'organiser les besoins utilisateur d'abord sous forme de listes distinguant besoins fonctionnels et non fonctionnels, avant de produire des cas d'utilisation.

Wisdom propose une légère variation de cette approche : les tâches utilisateur sont représentées par un diagramme d'activité intégrant des « cas d'utilisation essentiels » intégrant les réponses du système. Toutefois, à ce niveau de description, le détail des modalités employées par l'utilisateur ou le système (par ex., l'utilisation d'un clavier, l'affichage sur un écran, l'entrée d'un code PIN etc.) ne sont pas exprimés.

Hormis RUP et UCD Agile, ces méthodes respectent toutes les recommandations issues des travaux de Constantine [29].

Notons que les représentations des besoins utilisateur les plus structurées (par exemple, les flots de tâches Wisdom) facilitent les spécifications des besoins. En revanche, leur complexité ou leur manque de lisibilité auprès des non-experts peut freiner la communication avec utilisateurs et décideurs.

POUR RÉSUMER

La valeur attendue par l'utilisateur doit être représentée au cours de la phase de spécification, sous une forme structurée, à haut niveau d'abstraction (i.e., il s'agit de représenter les buts plutôt que les moyens). Les réponses du système peuvent également être décrites, à condition que celles-ci ne posent pas d'hypothèse quant aux modalités employées.

2.3.3.2 Scenarii et prototypes, modèles communicationnels universels

En ce qui concerne les prototypes ainsi que les scenarii, ceux-ci sont quasi unanimement utilisés, aussi bien pour décrire l'espace métier que l'espace interactionnel. Les prototypes apparaissent comme des supports de prédilection des échanges avec les utilisateurs (les six méthodes les utilisent), dans le contexte de la validation des spécifications ou de la conception. Les scenarii sont également largement utilisés (trois méthodes sur six), généralement comme des instances de cas d'utilisation.

POUR RÉSUMER

Les prototypes et scenarii sont des modèles adaptés à la communication avec les décideurs et/ou utilisateurs. Certains diagrammes UML peuvent remplir ce rôle s'ils sont accessibles sans connaissance préalable du langage.

2.3.3.3 Langages de modélisation

Considérant la représentation du métier, nous constatons sans surprise que le langage UML est très largement adopté. La représentation de l'interaction présente, quant à elle, des disparités sur la forme.

En effet, s'il existe un consensus quant à la nécessité de modéliser les tâches de l'utilisateur en amont du processus de développement de l'IHM – excepté RUP et UCD Agile, qui éludent cette pratique –, nous avons toutefois relevé plusieurs types de mise en œuvre :

- la méthode UPi utilise le formalisme CTT tel quel, comme raffinement des cas d'utilisation ;
- les méthodes Wisdom et User Engineering adoptent toutes deux une représentation des tâches utilisateur à l'aide de diagrammes de classes. Wisdom effectue ainsi une traduction du langage CTT vers le formalisme UML, afin de rendre celui-ci accessible aux concepteurs issus du GL ainsi qu'aux outils de conception classiques. User Engineering, en revanche, présente une modélisation des tâches utilisateurs comme un ensemble d'étapes permettant d'atteindre un but – un but étant une valorisation apportée par l'application à l'utilisateur, en rapport direct avec les buts définis par l'organisation. À ce titre, cette modélisation des tâches utilisateur est assez proche des représentations sous forme de cas d'utilisation ; il serait d'ailleurs tentant de la considérer comme une réorganisation du formalisme des cas d'utilisation ;

- la méthode Diane+ propose une représentation des tâches utilisateur très proche de CTT : la représentation est hiérarchique et des opérateurs permettent de définir l'enchaînement des tâches à un même niveau d'abstraction. Toutefois, le formalisme Diane+ fixe le périmètre d'utilisation aux tâches abstraites, c'est-à-dire non liées aux dispositifs ou actions physiques de l'utilisateur, à l'inverse de CTT qui permet de détailler les actions physiques de l'utilisateur et les réactions du système. En revanche, de même que pour le processus Diane+, aucune indication n'est donnée quant à l'intégration de cette modélisation avec les modèles Merise classiques.

Pour résumer

Si le langage UML reste le formalisme de prédilection pour la description de l'espace métier, la représentation des tâches utilisateur est centrale pour la description de l'espace interactionnel.
Par ailleurs, les représentations unifiées des modèles GL et IHM (par exemple, à l'aide du langage UML) facilitent la mise en relation des deux espaces conceptuels.

2.3.3.4 Modèles isolés et practiciens solitaires

UPi est la seule méthode à utiliser un langage issu du domaine de l'IHM : CTT. Les modèles CTT sont présentés comme des raffinements des cas d'utilisation, utilisés par les spécialistes IHM et les ergonomes pour la sélection et l'organisation des patrons d'utilisabilité constituant l'interface homme-machine. Aucune précision n'est donnée, en revanche, sur le processus de transfert de connaissances entre les spécialistes GL responsables des cas d'utilisation et les spécialistes IHM.

De même, Wisdom et User Engineering intègrent des activités de mise en correspondance de l'architecture fonctionnelle du système avec l'architecture de l'interface homme-machine. Ces deux méthodes s'appuient sur le langage UML pour modéliser l'ensemble de l'espace interactionnel (*« l'utilisation inter-disciplinaire d'UML comme* lingua franca *de l'équipe [de développement] est l'une des clés du transfert cohérent des connaissances [entre membres de l'équipe] »* [14] [97]). Toutefois, les détails de ces mises en correspondance ne sont pas explicités.

Pour résumer

La lacune constatée quant aux supports méthodologiques pour la collaboration se retrouve dans la rareté des modèles et l'absence de processus pour la collaboration entre les domaines du GL et de l'IHM.

14. *"The cross-disciplinary use of UML as the* lingua franca *of the team is one of the keys to coherent transfer of knowledge"*

2.3.4 Analyse des pratiques d'instrumentation des méthodes

Nous avons constaté au cours de notre étude que les méthodes les plus adoptées recommandent voire fournissent des outils d'aide à la conception. À l'inverse, DIANE+, qui ne fournit à notre connaissance aucun outil de modélisation pour sa notation ou bien en support de sa démarche, n'a ainsi pas été utilisé en-dehors du cadre scientifique. Au contraire, Wisdom fait état d'utilisations au sein d'entreprises de petite taille, initialement dépourvues de méthodologie de développement explicite. Ce transfert vers l'industrie s'est avéré concluant grâce à l'utilisation du langage UML, largement répandu dans le monde industriel : l'adoption de cette nouvelle méthodologie n'a ainsi pas remis en cause les outils déjà utilisés par les équipes de développement.

De même, la méthode UPi, qui a recours à des outils instrumentant les activités impliquant des modèles (CTT) ou des processus systématiques (choix des patrons d'utilisabilité), a été utilisé avec succès dans au moins un projet gouvernemental d'envergure.

Nous rapprochons ce phénomène des recommandations des rapports CHAOS de 1999 et 2001 [112, 113] selon lesquelles le développement doit être accompagné d'un environnement de conception efficace et adapté. Cette préconisation est d'ailleurs soulignée dans User Engineering (*« User Engineering suggère que le plus d'information possible soit stockée dans des outils d'aide à la conception »* [15]) [97]. Roberts insiste sur le fait que les collaborations au sein de l'équipe de développement doivent s'appuyer lourdement sur de tels systèmes informatisés. Il préconise notamment de tracer l'évolution des produits de la méthode à l'aide d'outils communs, ou tout du moins de permettre le regroupement de différents niveaux de raffinement des produits de la méthode.

En effet, le temps investi dans la description des différents produits requis par les méthodologies – modèles formels, scenarii, prototypes etc. – l'est toujours au détriment de l'implémentation proprement dite. Par conséquent, chaque produit doit être 1) réalisé dans un temps minimum ; 2) traçable (i.e., les liens entre produits doivent être explicites, et les produits eux-mêmes doivent donc être documentés) [97, 113]. Ce besoin est d'autant plus saillant lorsqu'il s'agit de décrire des modèles formels ou semi-formels (tels les modèles UML), dont l'élaboration est contraignante d'une part, et qui ne peuvent pas être utilisés comme modèles communicationnels d'autre part, justement à cause de leur complexité.

POUR RÉSUMER

L'instrumentation constitue un facteur central dans l'acceptation et l'efficacité des méthodes. Il est préférable de fournir un outillage adapté pour l'ensemble des tâches de développement, ainsi que pour les tâches transversales (organisation, etc.). D'autre part, l'outillage doit permettre de tracer l'évolution des produits ainsi que les collaborations.

15. *"User Engineering suggests that as much information as possible is held in [CASE] tools."*

2.4 Synthèse

Nous avons présenté dans ce chapitre un état de l'art des méthodes mêlant conception GL et IHM. Nous avons ainsi pu constater que ce domaine d'étude, marqué par quelques convergences, mais surtout ses contrastes, est encore en construction.

En considérant les processus, outillages, modèles et langages des méthodes RUP, Diane+, UPi, Ovid/User Engineering, Wisdom et UCD Agile, nous avons pu identifier un ensemble de pratiques récurrentes :

- le cycle de conception est préférablement itératif, incrémental, et ses activités sont parallélisées dès que possible ;
- l'implication des utilisateurs dans le développement est primordiale, notamment pour la capture de leurs besoins ;
- la capture des besoins utilisateurs doit se focaliser sur les besoins essentiels de l'utilisateur, formalisés sous forme de modèles communicationnels structurés et à haut niveau d'abstraction par l'équipe de développement, puis validés par l'utilisateur ;
- les utilisateurs et décideurs doivent pouvoir évaluer régulièrement les fonctionnalités et l'interface du système développé ; à défaut de présenter une version complète du système, les scenarii et prototypes (éventuellement, certains modèles UML simples) sont des modèles adaptés à cette tâche ;
- il est possible de réaliser les spécifications GL et IHM en parallèle, puis de rassembler ces spécifications au cours de la phase d'analyse ;
- les modèles de tâches utilisateurs sont indispensables à la spécification de l'IHM ;
- les représentations unifiées des modèles GL et IHM, par exemple avec UML et à condition que les modèles soient de faible complexité, facilitent la mise en relation des espaces conceptuels ;
- il est nécessaire de fournir un support méthodologique pour les collaborations, en particulier entre les domaines du GL et de l'IHM ;
- l'outillage est un facteur essentiel de l'utilisabilité des méthodes de développement.

Considérant les besoins méthodologiques de conception des systèmes de réalité mixte, nous constatons de nombreux points de convergence avec les pratiques des méthodes intégrant pratiques GL et IHM classiques. En particulier, l'aspect itératif des cycles de conception, l'implication des utilisateurs, l'utilisation des modèles IHM, l'implication des spécialistes IHM et des ergonomes, et la mise en œuvre de supports méthodologiques pour les collaborations sont des besoins pour la conception communs aux deux approches. Par la suite, nous déduirons de ces besoins un compendium de principes fondateurs d'une méthode de développement des systèmes de réalité mixte.

*

* *

Le chapitre suivant présente d'abord la méthode Symphony, issue du domaine du Génie Logiciel, et sur laquelle nous appuyons nos contributions.

CHAPITRE 3

La méthode Symphony, sa formalisation et son instrumentation

C'est dans les vieilles marmites qu'on fait les meilleures soupes.

PROVERBE POPULAIRE

LES TRAVAUX que nous avons étudié précédemment et qui ont eu un écho dans l'industrie sont construits sur des démarches issues des communautés GL et SI. Nous adoptons la même approche, ce qui nous permet de capitaliser sur la maturité et la maîtrise d'une méthode, dont les apports et les défauts sont bien connus. C'est dans cette optique que notre choix s'est porté sur la méthode Symphony.

Symphony est une méthode de développement centrée les composants métier, originellement développée au sein de la société Umanis. Umanis axe ses compétences sur *« la Business Intelligence, [la] Gestion de la Relation Client (CRM) et [l'] E-business »* [1]. Dans cette perspective, la méthode Symphony est destinée à *« la mise en œuvre de projets J2EE d'envergure »* [2].

Depuis plusieurs années, la méthode est au centre de plusieurs collaborations entre Umanis et l'équipe SIGMA du LIG. Des projets ambitieux (refonte du système d'information du service après-vente de l'enseigne Conforama [57] et migration d'une partie de l'intranet du système d'information hospitalier du CHU de Grenoble [63]) ont notablement permis de mettre en application les évolutions théoriques de la méthode.

Par ailleurs, Symphony adopte un cycle de développement dont l'une des focalisations est l'architecture technique. Cette propriété est particulièrement adaptée aux systèmes de

1. `http://www.groupeumanis.com/` – Date de consultation : 10 sept. 2009
2. `http://www.groupeumanis.com/index.php?option=com_content&task=view&id=72&Itemid=72` – Date de consultation : 10 sept. 2009

réalité mixte, dont l'architecture technique est, en général, sinon complexe, tout au moins inhabituelle.

De plus, les récentes évolutions de la méthode Symphony ont renforcé la traçabilité des artefacts du développement, permettant par exemple aux équipes de développement d'identifier clairement les points d'impact d'une modification du système lors des remaniements fonctionnels ou techniques, lesquels sont plus sensibles lorsqu'ils concernent un système de réalité mixte. Ce principe se pose *a contrario* des méthodes de développement agiles, qui minimisent – voire suppriment – la documentation des produits de la démarche. Ces dernières ne seront donc pas considérées – telles quelles, du moins – dans le cadre de ces travaux.

La pertinence de l'approche GL proposée par Symphony d'une part, l'expertise acquise au sein de l'équipe SIGMA d'autre part, justifiaient son choix comme medium d'intégration des pratiques GL et IHM.

Enfin, bien que la validation formelle de ce choix ne soit pas parmi nos objectifs, il convient de mentionner que Symphony a fait l'objet d'une comparaison – décrite par Oussalah et al. [90] – avec d'autres méthodes telles que Catalysis [44], RUP [72], Select Perspective [103] et 2TUP de la société Valtech.

À l'issue de cette étude, Symphony s'est distinguée grâce aux caractéristiques suivantes :

- elle intègre une approche fonctionnelle centrée sur les composants métier, dans une perspective de réutilisation et de réutilisabilité ;
- les composants métier sont identifiés de manière systèmatique lors de la spécification des besoins ;
- la traçabilité des choix est garantie tout au long du processus de développement ;
- elle intègre un modèle de composants métier tripartite original : les Objets Métier.

Évolution de la méthode Symphony

La démarche que nous présentons dans ce chapitre est distincte de celle utilisée au sein d'Umanis. Elle est en effet le fruit de plusieurs évolutions visant à en améliorer différents aspects. Ainsi, les travaux de Jausseran [63] et Hassine [58, 57] ont notablement intégré dans la méthode :

- un renforcement des phases métier, concrétisé par la description d'une phase d'étude préalable, d'une part, et une focalisation sur les processus métier lors de la spécification des besoins utilisateur, d'autre part ;
- l'identification de composants dès l'expression des besoins, à l'aide du concept d'Objet Métier ;
- la réutilisabilité des Objets Métier, en proposant la notion de « Composant Métier », construit *par* et *pour* la réutilisation ;
- la formalisation des différentes phases et activités du développement sous forme de patrons ;
- l'instrumentation de la méthodologie à l'aide d'un atelier pour la gestion et l'application de patrons (AGAP).

Juras et al. [66] ont par la suite envisagé l'implication de spécialistes issus de différents domaines du développement logiciel et décrit différents mécanismes de collaboration entre ceux-ci, dans la perspective d'intégrer *in fine* les pratiques de l'IHM et du GL.

3.1 Cycle de développement

Le cycle de développement de Symphony est caractérisé par un processus de développement dit « en Y », dont les principes et l'organisation emprunte aux méthodes Merise/2 et RUP.

3.1.1 Processus de développement en Y

Comme tout processus de développement, Symphony a pour objectif de spécifier les différentes phases d'un projet et de définir les tâches de chacun des intervenants du développement. Il s'agit d'une démarche itérative et incrémentale, centrée sur l'architecture, les processus métier et la gestion du risque, et dont les entités de premier ordre sont les processus métier (représentés sous la forme de cas d'utilisation) et les objets métier/composants métier. Cette démarche a adopté dès sa première version un modèle de cycle de vie en Y.

À chaque processus métier identifié lors d'une phase d'étude préalable correspond une instance du cycle complet de développement en Y. L'ensemble des processus métier est donc développé au cours de multiples instances et itérations du cycle de développement, selon le modèle en flocons proposé par J.-P. Giraudin [53] et présenté en figure 3.1. Cette approche reprend celle initiée par B.W Boehm [14], décrite précédemment.

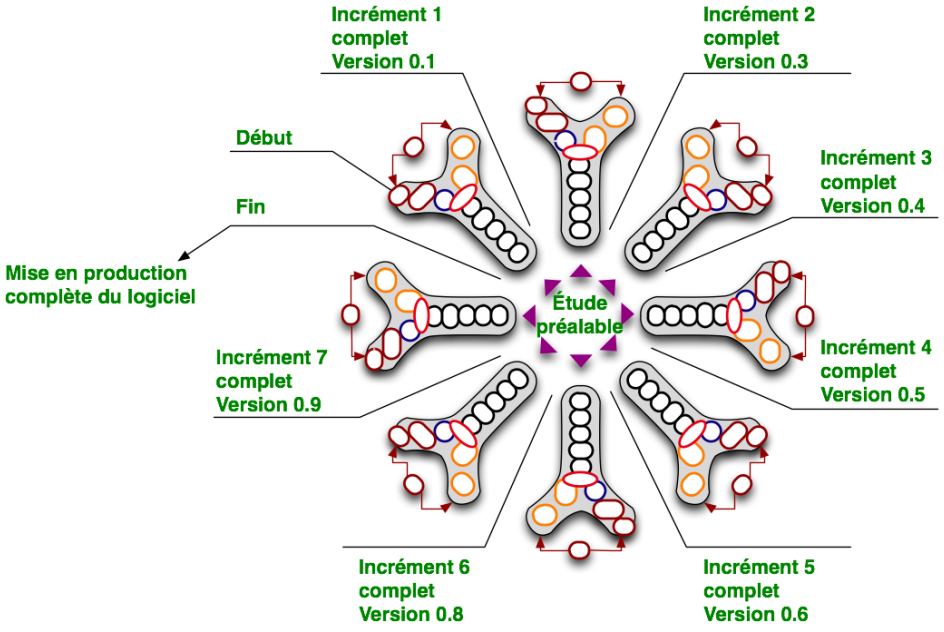


Figure 3.1 – Succession d'incréments de développement logiciel, selon le modèle en flocon de J.-P. Giraudin [53]

La figure 3.2 illustre le cycle de vie en Y tel qu'il a été décrit à l'issue des travaux de Jausseran et Hassine[63, 57].

Le développement d'une application doit prendre en considération deux aspects fondamentaux et complémentaires :

- Les fonctions attendues de l'application, censées répondre aux besoins du métier de l'entreprise,

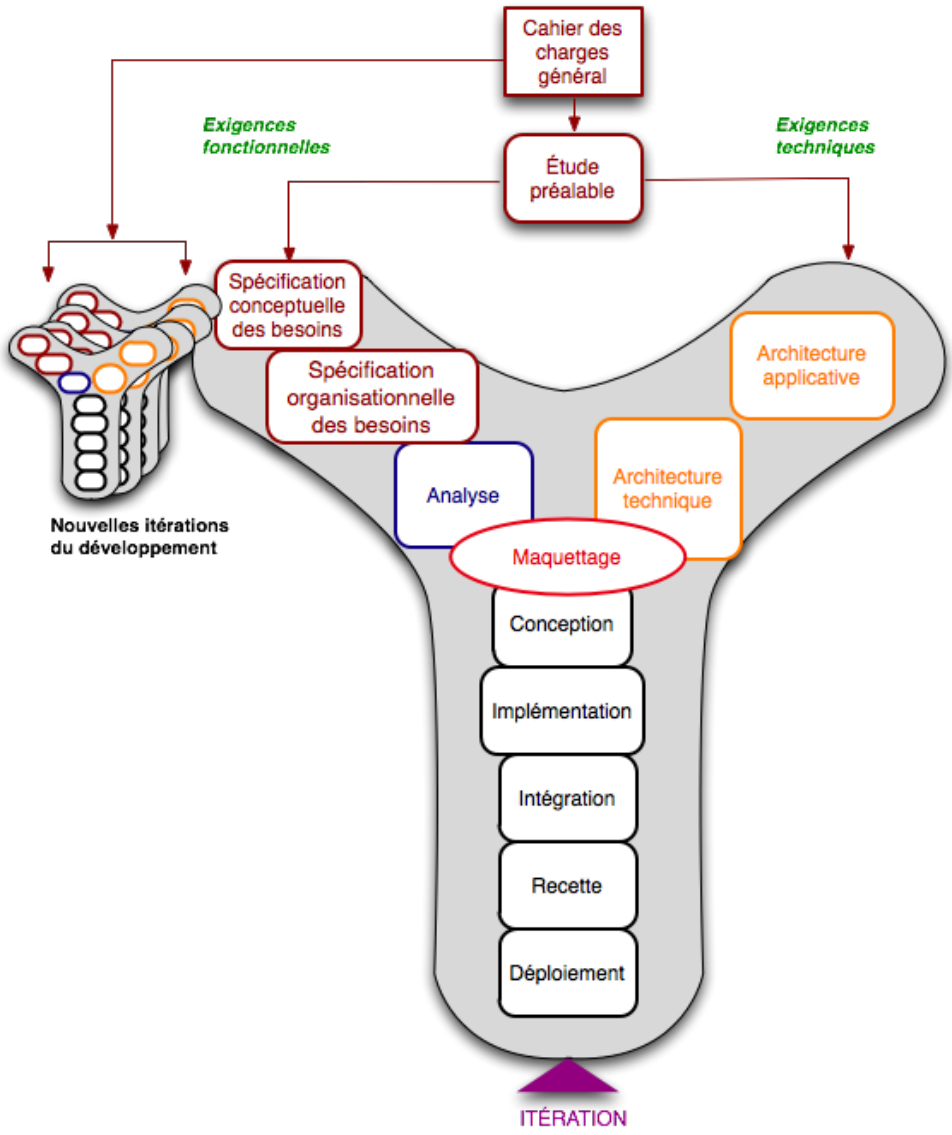


FIGURE 3.2 – Représentation du cycle de développement Symphony, à l'issue des travaux de Jausseran et Hassine[63, 57]

- Les contraintes opérationnelles de l'application, telles que les contraintes matérielles logicielles, la qualité de service en terme de temps de réponse, la tolérance aux pannes etc.

Symphony sépare l'étude des besoins fonctionnels de celle des besoins techniques et ce dès le début du cycle de développement. Cette approche permet une meilleure analyse du problème et des risques, mais aussi une meilleure réutilisation de l'existant. Menées en parallèle, ces deux activités se déroulent néanmoins en étroite collaboration afin d'en assurer la cohérence. De plus, les instances du développement, pour chaque processus métier, sont pilotées par les résultats de la phase d'étude préalable.

Le cycle de développement de la méthode Symphony est ainsi organisé comme suit :

- l'**étude préalable**, dont les objectifs sont de réaliser une modélisation des processus métier existant, sous forme de cas d'utilisation, et d'identifier les acteurs impliqués dans l'utilisation du produit final ;
- la **branche fonctionnelle (gauche)** correspond à la traditionnelle analyse du métier, ainsi qu'à la modélisation des besoins utilisateurs, indépendamment des caractères techniques de l'application ;
- la **branche technique (droite)** traite des aspects architecturaux de l'application, aussi bien en termes logiciels que matériels. Cette branche regroupe également les contraintes techniques (c'est-à-dire, non fonctionnelles), telles que la sécurité, l'équilibrage de charge, la persistance… ;
- la **branche centrale** intègre les modèles d'analyse de la branche gauche et l'architecture applicative définie dans la branche droite dans un modèle de conception. Celui raffine les composants jusqu'au niveau du code d'implémentation.

Dans cette version de la méthode, la description de l'IHM est concentrée dans une activité de maquettage, posée à la jonction des branches fonctionnelle et technique.

3.1.2 Une méthode héritière de Merise/2

Les phases les plus en amont de la méthode Symphony, à savoir l'étude préalable et la spécification conceptuelle des besoins, empruntent leurs objectifs à la méthode Merise/2. Cette dernière est en effet caractérisée par une étude conceptuelle de l'organisation et du métier de l'entreprise, en préalable à toute activité de développement informatique [92]. Cette étude doit permettre notamment de déterminer à la fois les processus métier essentiels de l'entreprise, des objets métier autours desquels ils s'organisent, et les acteurs externes du métier impliqués dans la réalisation du processus. Un processus métier est défini par Kettani et al. [67] comme :

> *« l'ensemble des activités internes d'un métier dont l'objectif est de fournir un résultat observable et mesurable pour un utilisateur individuel du métier »*

Les acteurs externes du processus métier peuvent être définis comme les entités (personnes ou systèmes) extérieures au système (ou à l'entreprise) interagissant avec celle-ci [57].

Dans le contexte d'une entreprise de gestion immobilière, la gestion des états des lieux constitue un processus métier à part entière, dont le document d'état des lieux est l'objet métier principal. Le tableau 3.1 présente un extrait de la description en langue naturelle du processus métier « Gestion des états des lieux » : les éléments de texte en gras désignent les

acteurs du processus métier ; les éléments en italique signalent les événements principaux intervenant au cours du processus métier ; les éléments en gras et italique présentent les ressources (objets métier) principales du processus métier. Ce processus métier implique plusieurs acteurs externes : les locataires entrants et sortants, et le propriétaire du logement.

Tableau 3.1 – Description du processus métier « Gestion d'un état des lieux »

- L'**agent immobilier (acteur interne)** *organise un rendez-vous (événement)* entre le **locataire (acteur externe)** (sortant dans le cas d'un état des lieux de sortie, entrant dans le cas d'un état des lieux d'entrée) et l'**expert état des lieux (acteur interne)** (salarié de l'agence immobilière, huissier, sous-traitant, etc.), choisi par l'agence immobilière chargée de réaliser l'***état des lieux (ressource)*** ;
- l'**agent immobilier** *fournit* à l'**expert état des lieux (EdL)** les informations nécessaires à la bonne réalisation de l'***EdL*** (telles des fiches de renseignements à remplir, les états des lieux précédents, si nécessaire, la liste des artefacts à récupérer etc.) ;
- l'**expert EdL** *réalise* l'***EdL***, en présence du **locataire**, qui peut émettre des objections quant aux constats de l'**expert EdL** ;
- en cas de désaccord sur le contenu de l'***EdL***, une *comparaison est faite* par l'**expert EdL**, avec l'***EdL précédent*** si celui-ci est disponible ;
- les artefacts du logement (clefs diverses, par exemple) sont soit *récupérés* (EdL sortant), soit *remis* (EdL entrant) au **locataire**. L'***EdL*** est signé par l'**expert EdL** et le **locataire** (sauf en cas de désaccord majeur) ;
- l'**expert EdL** *transmet l'état des lieux* à l'**agent immobilier**, qui *met le dossier du logement et de location à jour* ;
- si nécessaire, l'**agent immobilier** contacte le **propriétaire (acteur externe)** pour discuter d'éventuelles réparations à apporter, ou simplement des usures ou dégradations constatées, et envisager une remise à niveau du logement.

Cette approche de la conception permet notamment d'évaluer la pertinence du projet de développement logiciel, et de déterminer un découpage systématique de l'application en blocs fonctionnels indépendants afin d'en planifier le développement. De plus, l'identification précoce des acteurs externes permet d'envisager, avant même la mise en œuvre du développement, l'intégration du futur système dans son écosystème d'information.

3.1.3 Une méthode héritière du RUP

D'un point de vue général, Symphony utilise exclusivement le langage UML pour réaliser les différents artefacts du développement. Il est par conséquent possible pour les équipes de développement de capitaliser sur l'outillage varié et exhaustif supportant ce standard. De même, Symphony propose également un cycle de développement incrémental et itératif.

D'un point de vue plus spécifique, les phases aval de la branche fonctionnelle de Symphony (spécification organisationnelle des besoins puis analyse) reprennent des concepts propres au Rational Unified Process. La spécification organisationnelle des besoins préconise ainsi la traduction de l'analyse des processus métier en cas d'utilisation, d'où sont extraits les objets métier du système. Hormis la structure spécifique des objets métier (c.f section suivante), ceux-ci sont par la suite classiquement raffinés en diagramme de classes et diagrammes de séquences lors de la phase d'analyse.

3.2 Le modèle d'Objet Symphony : modélisation de l'espace métier sous forme d'Objets Métier

À la suite des phases de spécification, le système est vu, tant au niveau conceptuel que logiciel, comme un assemblage d'Objets Métier (OM) indépendants et interconnectés. Cette pratique garantit une bonne modularité des spécifications et facilite leur réutilisation.

Le modèle métier de la démarche spécifie trois types d'OM. Aux deux catégories fondamentales d'OM Processus et Entité, Symphony ajoute les OM Données de référence :

- les OM « **Processus** » permettent de décrire un processus applicatif (par exemple, le processus de réalisation d'un état des lieux, incluant les règles applicatives relatives à la description d'un dommage) ;
- les OM « **Entité** » constituent la base des OM et décrivent les concepts métier (par exemple, le document d'état des lieux) ;
- les OM « **Données de référence** » représentant les données de référence, c'est-à-dire des structures de données de faible complexité, spécifiques au domaine métier (par exemple, la nomenclature des types de dommages relevés par l'agence immobilière).

Les OM sont d'abord identifiés et organisés au cours de la phase de spécification organisationnelle des besoins. Comme l'illustre la figure 3.3, les OM ont alors la forme de paquetages stéréotypés, liés entre eux par des relations de dépendance « utilise » (*use*) : un objet métier processus « Réaliser un état des lieux » (*Realize inventoy of fixtures*) utilise l'Objet Métier Entité principal « État des lieux » (*Inventory of fixtures*), lui même ayant recours à l'OM Entité « Dommage » (*Damage*).

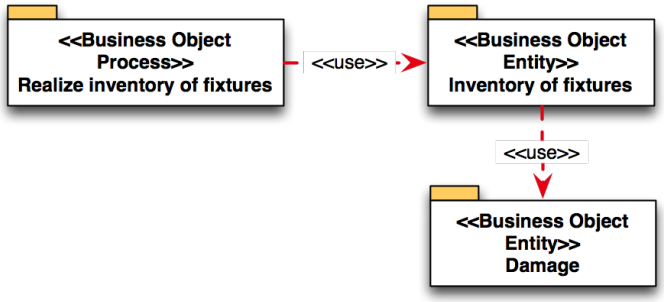


Figure 3.3 – Cartographie d'Objets Métier, de niveau spécifications

L'organisation des OM, réalisée à haut d'abstraction, est raffinée au cours de la phase d'analyse. Chaque OM est alors décrit sous la forme d'une structure UML tripartite, inspirée de la formalisation CRC (Classe – Responsabilité – Collaboration) [10], mettant en évidence les services offerts par l'OM, sa structure interne et les objets auxquels il est lié.

La figure 3.4 reprend les Objets Métier de l'application d'état des lieux, pour en décrire le raffinement de niveau analyse. La partie gauche du paquetage décrit les services fournis par l'objet, à l'aide d'une classe stéréotypée « Interface ». La partie centrale du paquetage décrit l'implémentation de ces services, à l'aide d'une classe stéréotypée « Maître » (*Master*), ainsi qu'une subdivision en concepts complémentaires, à l'aide de classes stéréotypée « Partie » (*Part*, non représentées dans la figure). Enfin, la partie droite (classes « Rôle ») décrit les services requis par l'objet pour garantir son comportement, de la part des Objets Symphony dont il dépend (c'est-à-dire qu'il existe une relation « utilise » entre ces objets). Les Objets

Métier « Données de référence » (*Business Data Reference*) font également leur apparition lors de la phase d'analyse.

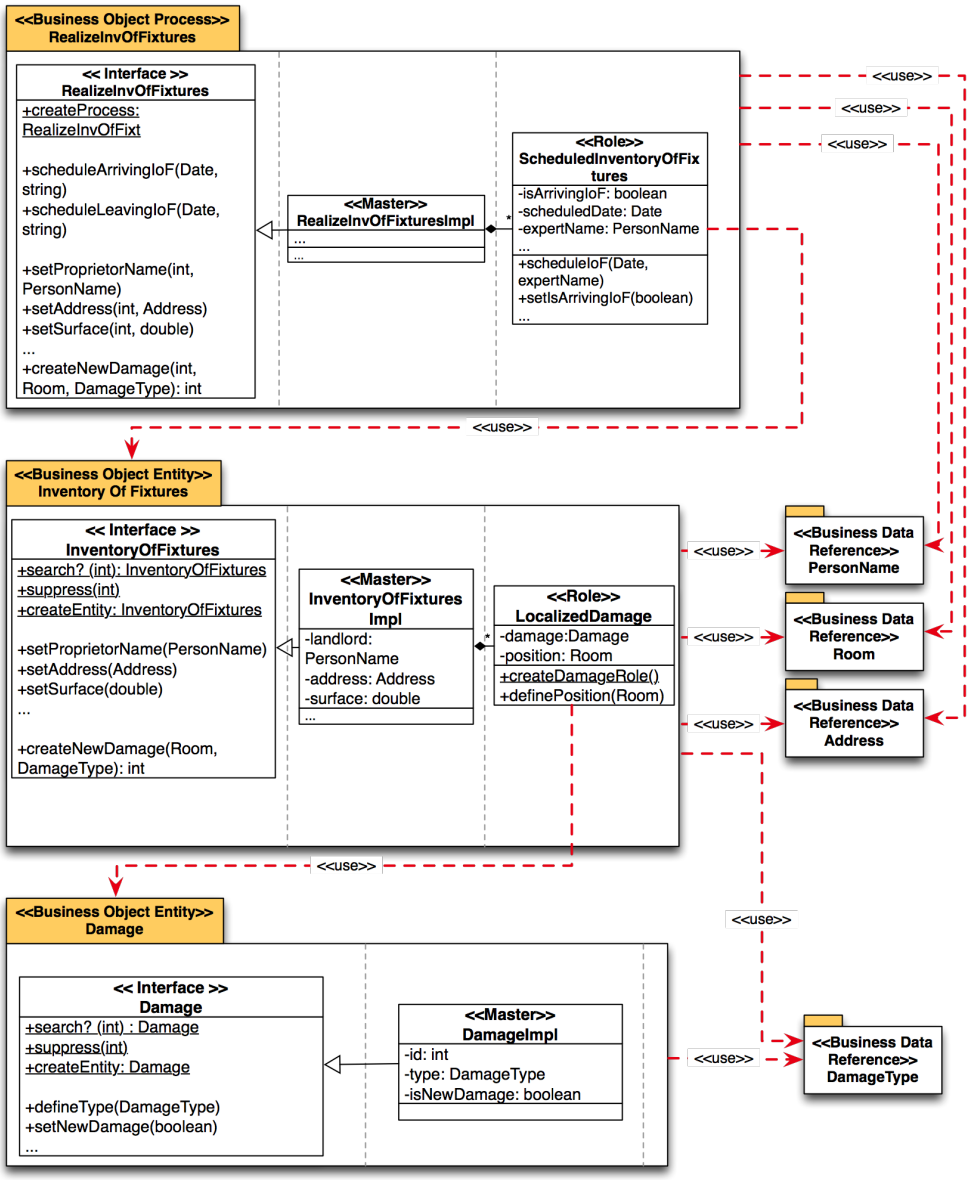


FIGURE 3.4 – Diagramme de classes d'Objets Métier, de niveau analyse

Dans les phases ultérieures (conception et implémentation) de la branche centrale du Y du cycle de développement, ces Objets Métier conceptuels sont progressivement transformés en OM logiciels par intégration des choix techniques spécifiés dans les objets et composants techniques.

De même que la démarche est basée sur les OM, elle l'est aussi sur les objets et composants techniques. Ces derniers représentent des composants non fonctionnels permettant de définir l'architecture technique de l'application et proposant des solutions à des problèmes techniques récurrents tels que la communication réseau, les connexions d'unités physiques, la persistance etc.

3.3 Artefacts et traçabilité

Les processus métier, organisés sous forme de cas d'utilisation et définis en amont de la méthode, constituent le point d'achoppement de la traçabilité des produits du développement. Chaque artefact conçu au cours du cycle en Y est en effet relié à un cas d'utilisation par une relation de dépendance parmi les trois types « est spécifié par », « est réalisé par » et « est implémenté par » [62]. Ces dernières raffinent la relation « traçabilité » (*trace*) du langage UML.

Les produits des phases de spécification et d'analyse sont liés aux cas d'utilisation par des relations « est spécifié par ». Les produits de la phase de conception et de la branche technique sont reliés aux cas d'utilisation par des relations « est réalisé par ». Enfin, les produits de la phase d'implémentation sont reliés aux cas d'utilisation par des relations « est implémenté par ».

Outre le suivi des produits de la conception, ces relations de traçabilité permettent également de répercuter les évolutions fonctionnelles et techniques sur les artefacts idoines, garantissant ainsi la maintenabilité du système développé.

3.4 Des patrons pour la documentation de la méthode

Les extensions récentes de la méthode Symphony ont notamment consisté à formaliser certains composants de la méthode, c'est-à-dire, le processus et les modèles, sous forme de patrons. Afin de capitaliser et de réutiliser des ensembles cohérents de connaissances, les patrons offrent une approche indépendantes des domaines modélisés. Ils constituent ainsi un moyen de documenter des méthodologies ou des architectures logicielles et permettent, dans le cas d'un développement en équipe, d'établir un vocabulaire commun.

3.4.1 Historique

La notion de patron est d'abord apparue dans le domaine de l'architecture des bâtiments, à la fin des années 70. Introduite par Christopher Alexander [1], cette approche a permis de mettre en évidence, à partir d'une analyse historique de l'architecture que, malgré l'absence de modèles préétablis et de règles rigoureuses, les constructions humaines obéissent à des préoccupations, dans l'espace et dans le temps, de nature récurrente. Cette prééminence des « motifs » dans la fabrication de systèmes complexes a été reconnue depuis dans de nombreuses disciplines.

Dans le domaine de l'informatique, c'est en 1987 que W. Cunningham et K. Beck [37] ont présenté les premiers patrons orientés sur la modélisation objet. Depuis, de nombreux travaux ont porté sur l'identification de patrons dans des domaines d'application variés et sur les différentes phases d'un processus de développement.

3.4.2 Définition

Alexander définit les patrons de la façon suivante :

> *« Chaque patron décrit à la fois un problème qui se manifeste constamment dans notre environnement et l'architecture de la solution à ce problème, de telle façon que vous puissiez utiliser cette solution des millions de fois sans jamais l'adapter deux fois de la même manière. »*

On distingue dans cette solution trois éléments essentiels à la caractérisation des patrons :

- Un patron intègre la description d'un **problème récurrent**,
- Un patron décrit une **solution générale éprouvée** (dans le domaine de l'informatique, un savoir-faire) à ce problème,
- Un patron inclut également des **moyens d'adaptation** de cette solution à des contextes spécifiques.

3.4.3 Classification

Il existe deux grandes catégories de patrons : les patrons processus et les patrons produit.

3.4.3.1 Patron processus

Introduits en 1995 par J. Coplien [32], les patrons processus se distinguent en patrons organisationnels – décrivant des démarches adoptées dans les organisations humaines –, et les patrons de développement logiciel guidant la mise en œuvre d'architectures logicielles, de technologies etc. Par la suite, et par commodité, nous assimilerons les patrons processus aux patrons organisationnels, le second type de patron n'étant pas abordé dans nos travaux.

Les patrons processus décrivent des solutions à des problèmes organisationnels récurrents sous la forme d'une démarche ou d'un fragment de démarche, dont les étapes peuvent elles-mêmes être décrites sous la forme de patrons processus. La solution formelle est généralement constituée d'un diagramme d'activité UML, sur lequel sont représentées la consommation ainsi que la production d'artefacts. Comme nous le verrons dans la suite de ce chapitre, le processus guidant la réalisation de la phase d'étude préalable de la méthode Symphony fait l'objet d'une formalisation sous la forme de patron processus.

3.4.3.2 Patron produit

Un patron produit est censé apporter une réponse sous la forme d'un ou plusieurs modèles, sans qu'il soit jugé pertinent de donner une démarche pour en décrire l'application. À l'instar des patrons processus, la littérature distingue différents types de patrons produit, tels que les patrons d'architecture logicielle ou de conception, décrits par Gamma et al. [52] ou les patrons décrivant les domaines métier [69]. La méthode Symphony décrit également les artefacts du cycle de développement sous la forme de patrons produits, telle la classification des acteurs externes et internes du système.

3.4.4 Systèmes de patrons

Quelle que soit leur portée (analyse, conception, implémentation…), les patrons sont regroupés en systèmes ou catalogues, dont l'ouvrage *« Design Patterns: Elements of Reusable*

Object-Oriented Software » de Gamma et al. [52] est un exemple typique.

Tout comme la notion de patron, le terme de système de patrons a été introduit par Alexander, de la façon suivante :

> *« Un système de patrons est une collection de patrons formant un vocabulaire qui permet de comprendre et de communiquer les idées. »*

Les patrons peuvent ainsi être classés par catégorie et étroitement liés entre eux par des problématiques similaires. Ces systèmes ont donc pour but de résoudre des problèmes variés et complexes, alors qu'un patron traite un problème spécifique.

Différentes approches ont traité la mise en relation des patrons constituant un système : certains systèmes de patrons [37, 52] utilisent des mots-clés pour rassembler les patrons selon différentes catégories, ou bien effectuent des références textuelles aux patrons du système : dans le système de Gamma et al., les patrons référencés – tous patrons produits d'architecture – sont regroupés dans une rubrique « Patrons apparentés » (*Related patterns*), créant ainsi des réseaux de relations non spécifiques. Les patrons décrits par C. Alexander [1] incluent également une rubrique « Contexte » décrivant les patrons prérequis, permettant ainsi de naviguer dans l'ensemble de la hiérarchie.

Par ailleurs, les systèmes de patrons tels que décrits à l'aide du formalisme P-Sigma [30], spécifiquement destinés à guider la mise en œuvre de démarches, incluent plusieurs types de relations de dépendance, telles que des relations d'*utilisation*, d'*alternative*, de *raffinement* et de *prérequis*, dont l'union forme un maillage de relations sémantiques entre les patrons du système.

3.4.5 Le formalisme P-Sigma

Les patrons de la méthode Symphony sont organisés selon un système de patrons défini par un formalisme composé de rubriques et de champs : P-Sigma [30]. Ce dernier formalisme a démontré sa forte capacité de description, d'organisation et de réutilisation des patrons dans le cadre du GL. Il permet essentiellement de décrire un patron selon trois parties :

1. **Interface** : contient les rubriques permettant l'identification du problème traité ainsi que l'identification, la classification du patron ainsi que les acteurs du système impliqués dans la réalisation du patron. Cette partie permet notamment de décrire la valeur ajoutée par l'application du patron (rubrique forces),
2. **Réalisation** : contient les rubriques décrivant la solution proposée par le patron, en terme de solution graphique et textuelle, des exemples de mise en œuvre de la solution, ainsi que les possibilités d'adaptation en fonction du contexte,
3. **Relations** : cette dernière rubrique permet de décrire d'une part, les patrons utilisés ainsi que les patrons dont le patron courant dépend : ces relations définissent la position du patron dans la méthode, en termes de niveau de granularité (phase, activité…) et d'enchaînement. Par ailleurs, les relations permettent de décrire les alternatives à l'application du patron courant (c'est-à-dire, les patrons répondant au même problème, mais mettant en avant d'autres types de valeur ajoutée), complémentant ainsi les capacités d'ouverture de la méthode.

La figure 3.5 présente une représentation simplifiée du formalisme P-Sigma, dans laquelle sont répertoriées toutes les rubriques utilisées pour la description de la méthode Symphony. Par ailleurs, le tableau 3.2 présente un extrait du patron processus « Phase d'étude préalable » de la version de Symphony décrite par E. Jausseran [63].

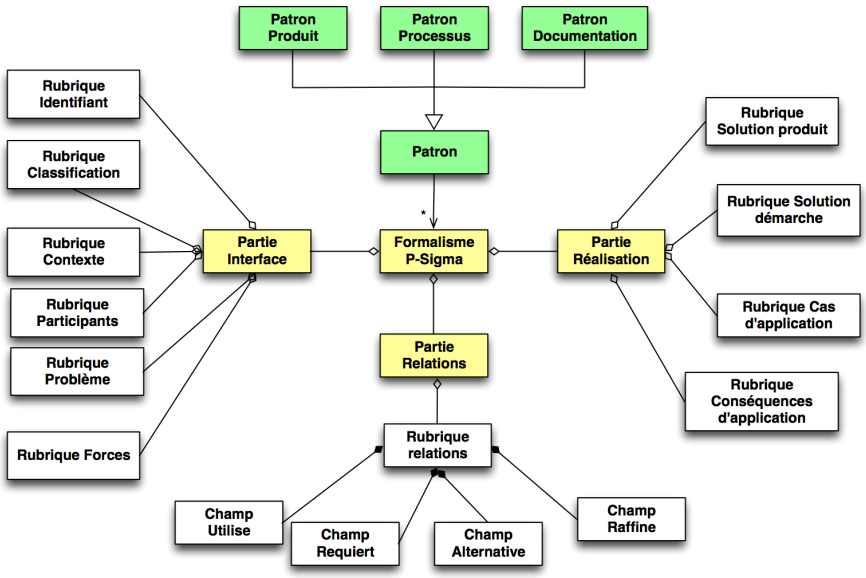


FIGURE 3.5 – Représentation du formalisme P-SIGMA

TABLEAU 3.2 – Patron processus de l'étude préalable de la méthode Symphony étendue, d'après E. Jausseran [63]

Interface

Identifiant

Phase d'étude préalable

Participants

- Maître d'ouvrage
- Responsable métier
- Analyste des processus métier

Contexte

{Modélisation métier ∧ Processus métier ∧ Processus ∧ Objet métier }

Problème

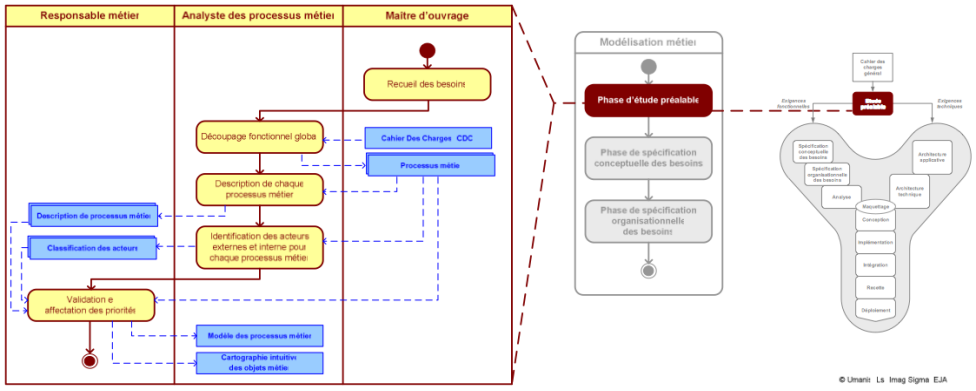
L'objectif de cette phase est un découpage fonctionnel du métier en vue d'identifier les processus métier (processus liés au métier de l'entreprise) et les différents acteurs (externes et internes) de chacun d'eux.

Forces

- Modèle des processus métier :
  - processus métier ;
  - diagramme UML des cas d'utilisation des processus métier ;
  - description des processus métier ;
  - classification des acteurs ;
- cartographie des objets métier (principaux) :
- itérations de développement par processus métier ordonnancées par priorité décroissante.

Réalisation

Démarche formelle



Démarche textuelle

La solution proposée par ce patron repose sur les activités suivantes :

- **Recueil des besoins** : le maître d'ouvrage rédige le recueil des besoins ;
- **Découpage fonctionnel global** : consiste en une recherche des processus métier. L'objectif est d'identifier et de spécifier les différents processus métier. Une vue d'ensemble du domaine est nécessaire pour une première modélisation des besoins du système final. Un processus métier est modélisé par un **cas d'utilisation**. Plusieurs acteurs peuvent être concernés, dont plus particulièrement l'acteur principal ;
- …

Chaque processus métier concerne intuitivement un objet métier principal. L'ensemble des processus métier dresse donc une cartographie implicite des objets métier.

Solution produit

- **Un livrable global** : il donne, d'une part, une vue récapitulative des résultats (sous forme de diagramme UML, de formulaires, …) obtenus lors de l'exécution des différentes activités, un **modèle des processus métier**, une **cartographie intuitive des objets métier**, et d'autre part une idée de l'avancement du projet ;
- …

3.4.6 L'outil AGAP

La méthode Symphony est actuellement décrite dans l'« atelier de gestion et d'application de patrons » (AGAP [31]). Cet outil polyvalent offre des solutions combinant les besoins de formalisation des méthodologies – sous forme de patrons – et de développement logiciel, domaine dont l'exigence est de pouvoir exploiter efficacement les données formalisées dans les systèmes de patrons. Ainsi, AGAP permet d'une part de définir des formalismes de patron, d'autre part de décrire des patrons à l'aide des formalismes entrés, et enfin de transformer les systèmes de patrons ainsi renseignés en sites web autonomes.

L'outil consiste en un site internet, dans lequel l'ingénieur de patrons peut créer, modifier, valider et visualiser un formalisme ou un système de patrons. Ces derniers permettent également d'explorer les différentes phases, activités et produits de la méthode en navigant suivant les différentes relations, exprimées sous forme de liens hypertextes, et d'effectuer des recherches textuelles.

Grâce aux guides méthodologiques générés par AGAP, les concepteurs vont pouvoir effectuer des recherches, naviguer en suivant les processus préconisés par les méthodes, identifier les étapes franchies et restantes et plannifier les opérations de modélisation et de conception à mener pour aboutir à des systèmes opérationnels.

Une capture d'écran d'un site web généré par AGAP est présenté en figure 3.6. On y retrouve l'essentiel des fonctionnalités facilitant la navigation dans les systèmes de patrons :

- Les fiches des patrons sont accessibles à partir du cadre latéral de l'outil,
- La représentation des solutions formelles (par ex. les diagrammes d'activité UML) peut inclure des zones sensibles HTML qui, une fois sélectionnées à l'aide de la souris, mènent via un lien hypertexte vers le patron correspondant,

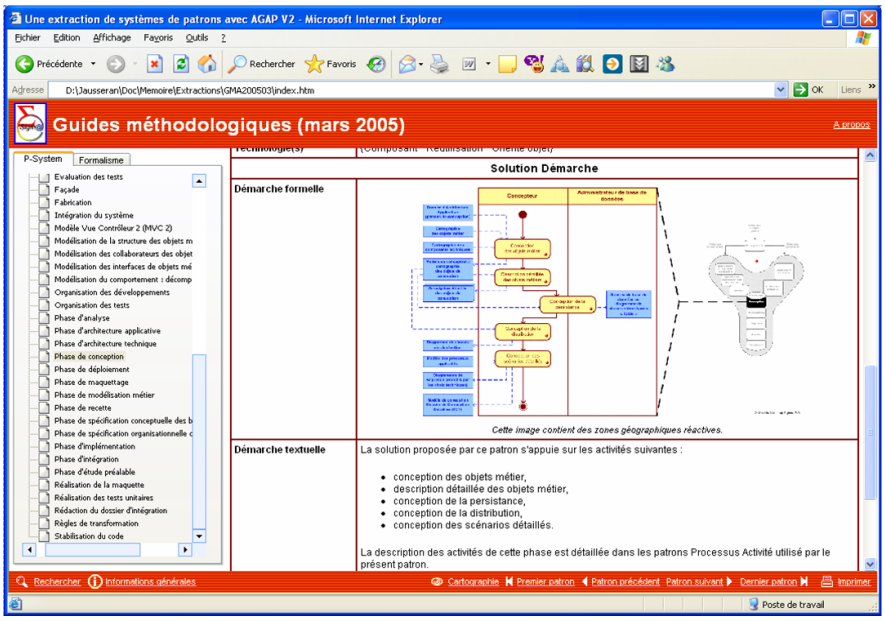


FIGURE 3.6 – Extrait de site web généré par l'outil AGAP

- Les champs formels sont représentés sous la forme de liens hypertextes. Leur sélection mène l'utilisateur à la fiche du patron correspondant,
- L'outil de navigation inclut également une interface de recherche par mots-clef.

3.5 Limites de la méthode

Sous la forme présentée ici, la méthode Symphony ne décrit le développement de l'IHM que de manière minimale : la phase de maquettage, située à la réunion des branches fonctionnelle et technique, consiste, comme son nom l'indique, à réaliser des maquettes du futur système. L'adéquation des maquettes avec les besoins utilisateur est validée par le responsable métier, après quoi celles-ci sont implémentées par l'analyste programmeur. Présentée sous cette forme, la méthode est clairement inadaptée au développement d'IHM riches ou de réalité mixte.

En particulier, la conception de l'IHM intervient tardivement dans le processus de conception, et les utilisateurs ne sont impliqués qu'indirectement dans la définition des besoins et la validation de la maquette. Par conséquent, la méthode Symphony compromet la capacité des développeurs à proposer des solutions cohérentes du point de vue de l'utilisabilité.

D'autre part, les modèles dits communicationnels ou collaborationnels sont de second ordre dans la méthode Symphony : les descriptions textuelles des processus métier, avec lesquelles l'équipe de développement peut communiquer avec les décideurs, sont présentés comme des compléments aux diagrammes de séquence. D'autre part, l'absence de collaborations entre domaines de compétence distincts nécessite l'instauration de modèles collaborationnels adaptés.

Quant à l'outillage de la méthode Symphony, nous avons constaté que l'outil AGAP mériterait d'être amélioré du point de vue de son ergonomie. Reprenant les critères de J. C. Bastien et

D. Scapin [7], nous estimons que les informations de guidage de l'utilisateur nécessitent d'être amélioré : il est difficile pour ce dernier d'identifier sa localisation au sein du système de patrons ; de plus, le cadre latéral de navigation présente l'ensemble des patrons par ordre alphabétique, ne reflétant pas la structure de la méthodologie. Dans une moindre mesure, la flexibilité et l'homogénéité/cohérence de AGAP sont perfectibles : l'ingénieur de patrons ne peut pas modifier un formalisme de patron une fois celui-ci validé, situation contraignant celui-ci à supprimer un système de patron complet dès lors que ce dernier requiert une modification de formalisme ; de plus, la prévisualisation des patrons proposée lors de l'édition des patrons ne correspond pas à l'apparence du site généré.

Enfin, l'outillage des produits de la méthode s'appuie sur les mêmes outils que ceux proposés pour RUP, tout particulièrement les outils de conception basés sur UML. Symphony propose une modélisation originale des concepts métier sous forme d'Objets Métier. Cette modélisation évolue au fil des phases du développement, de manière semi-systématique : il est donc envisageable d'automatiser cet aspect de la conception afin d'alléger la charge de travail des développeurs.

3.6 Synthèse

Nous avons présenté dans ce chapitre la méthode Symphony, que nous utiliserons comme support de nos contributions. Méthode éprouvée issue de la communauté du génie logiciel, elle est caractérisée par un cycle de développement original dit « en Y ». Ce dernier distingue en deux de conception distinctes l'analyse fonctionnelle de l'analyse technique, avant d'intégrer les modèles obtenues en vue de l'implémentation du système. De plus, Symphony propose la modélisation des concepts métier sous la forme d'Objets Métier, menant ainsi à une identification précoce de composants métier. Enfin, la méthode est documentée à l'aide de patrons processus et de patrons produits, et instrumentée par l'outil de gestion de patrons AGAP.

Les caractéristiques du processus Symphony sont résumées dans le tableau 3.3.

Tableau 3.3 – Grille d'analyse du processus Symphony originel

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	M		
Spécification des besoins	M | G		
Analyse	G		
Conception	G	G	

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome

Les caractéristiques des modèles utilisés dans la méthode Symphony sont exposées dans le tableau 3.4.

Les caractéristiques des outils employés dans la méthode Symphony sont décrits dans le

Tableau 3.4 – Grille d'analyse des modèles Symphony originelle

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL	UML		
Modèle IHM			
Scénario	(Cas d'utilisation)		
Prototype		Maquette	

Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

tableau 3.5.

Tableau 3.5 – Grille d'analyse de l'instrumentation de Symphony originelle

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
Outil de gestion de patrons AGAP	Outils d'aide à la conception UML	–

Malgré ses nombreuses qualité, nous avons également identifié un certain nombre de limites de la méthode Symphony, notamment la place secondaire tenue par la conception de l'interaction homme-machine, que nous employons à corriger dans les chapitres suivants.

*

* *

Dans la partie suivante, nous décrivons un ensemble de principes qui ont pour ambition de corriger les lacunes de Symphony, et permettre ainsi son utilisation dans le cadre du développement des interfaces classiques et des systèmes de réalité mixte.

Deuxième partie

Une méthode de développement des systèmes de réalité mixte

CHAPITRE 4

Principes pour une méthodologie de conception des systèmes de réalité mixte

> Anyone can make the simple complicated. Creativity is making the complicated simple.
>
> Charles Mingus

L'analyse proposée au chapitre 2 nous permet de définir un ensemble de pratiques vertueuses à intégrer dans une méthode de développement. En synthèse de ce chapitre (p. 85), nous avons rapproché ces bonnes pratiques des besoins méthodologiques spécifiques des systèmes de réalité mixte. Il s'agit à présent de décrire les principes sur lesquels nous basons notre reconstruction de la méthode Symphony, dans laquelle nous devons intégrer les préoccupations, modèles et pratiques de l'IHM.

Reprenant l'analyse menée au chapitre 2, nous proposons d'établir nos contributions méthodologiques sur une évolution des caractéristiques générales de la méthode Symphony :

- nous reprenons les mêmes entités de premier ordre que la méthode Symphony, à savoir les *processus métier* organisés sous forme de *cas d'utilisation* et les *objets métier*/*composants métier*, que nous renommons pour les besoins de nos contributions *Objets Symphony* et *Composants Symphony* ;
- nous centrons notre méthodologie sur les *besoins essentiels de l'utilisateur* (dans l'optique de Constantine [29], c'est-à-dire « centrée sur l'usage »), les processus métier et la gestion des risques ;
- à l'instar de Symphony, notre méthode est *itérative* et *incrémentale* ; les activités sont *parallèlisées*, et nous proposons en particulier un mécanisme de *gestion d'évolution des artefacts* du développement, basée sur la traçabilité.

Comme nous l'avons vu précédemment, une méthode est définie par quatre composants indissociables et complémentaires [96] : les *modèles* représentant une vue du système, les *langages* (permettant de construire les modèles), une (des) *démarche(s)* (fil conducteur) guidant les activités de conception, et enfin des *outils ou techniques* permettant et facilitant l'utilisation et la mise en œuvre des modèles, démarches et langages. En prenant en compte ces quatre composants, comment construire une méthode de conception pour les systèmes de réalité mixte ? Sur quels composants baser la méthode ?

1. Une **intégration harmonieuse des pratiques GL et IHM** : les méthodes que nous avons étudiées supposent que tous les acteurs partagent la même culture du développement logiciel, voire une culture issue du génie logiciel. Toutefois, l'intégration des pratiques du GL et de l'IHM doit impliquer des experts issus des deux domaines, sans révolutionner les pratiques – éprouvées – de l'un ou l'autre domaine. Au contraire, les outils, modèles et démarches utilisés dans les deux domaines doivent être préservés.
2. Des **activités collaboratives de conception** : afin de permettre aux spécialistes IHM, aux ergonomes, aux spécialistes du génie logiciel et aux experts métier de travailler de concert, nous devons élaborer des activités de développement collaboratives, qui permettrons la synchronisation sur des objectifs ou des modèles communs.
3. Des **modèles traçables et cohérents pour la collaboration et la communication** : les méthodes telles que Wisdom ou User Engineering décrivent les spécifications des besoins utilisateur dans des formalismes censés permettre une communication non-ambiguë avec les utilisateurs et les décideurs. De plus, afin de permettre des collaborations entre experts GL et IHM, des pratiques de modélisation consensuelles et non invasives doivent être établies. Enfin, considérant les risques induits par le développement des systèmes de réalité mixte, il est nécessaire de garantir une traçabilité ainsi qu'une cohérence entre les modèles, afin de permettre un pilotage précis de la méthode.
4. Un **outillage omniprésent** : suivant les recommandations de Roberts [97] et des rapports CHAOS [111], ainsi que les pratiques du RUP [72] et de Wisdom [83], notre méthodologie doit être instrumentée par une large gamme d'outils, depuis la plannification et les guides méthodologiques jusqu'à la génération du code. D'autre part, ces outils doivent permettre la mise en œuvre des principes mentionnés ci-dessus.

Ces principes sont détaillés ci-dessous.

4.1 Une intégration harmonieuse des pratiques GL et IHM

Ce premier principe implique tout d'abord d'identifier et de définir les *acteurs* intervenant dans la méthode, ainsi que le choix des *processus de conception* et des *modèles* qui leurs sont associés.

4.1.1 Les acteurs

Nous reprenons, pour la description de notre méthode, la classification utilisée pour l'analyse des méthodes GL/IHM au chapitre 2. Les rôles fonctionnels considérés sont donc : « spécia-

liste GL », « spécialiste IHM », « spécialiste métier » et « ergonome », dont nous rappelons les responsabilités :

- le **spécialiste métier** est un référent de l'équipe de développement pour tout ce qui concerne le métier à informatiser ; il porte également la responsabilité des découpages conceptuels du métier (par les flux, par les produits, etc.) ;
- le **spécialiste GL** possède des compétences en conception de systèmes logiciels, couvrant l'ensemble des domaines techniques liés à la conception et à l'implémentation d'applications (hors compétences IHM) ;
- l'**ergonome** dispose d'une formation en psychologie sociale, anthropologie ou sociologie, mais ne possède pas de connaissances « fondamentales » en informatique, et axe sa pratique sur l'évaluation, le conseil, etc., en interaction homme-machine ;
- le **spécialiste IHM** est un informaticien spécialisé dans la conception d'IHM du point de vue logiciel, possédant des notions d'ergonomie logicielle.

4.1.2 Les processus et les modèles

Nous déterminons ci-dessous les processus et modèles mis en œuvre, parmi les pratiques GL de la méthode Symphony originelle, les pratiques IHM de conception des systèmes de réalité mixte, ainsi que la démarche d'intégration des deux pratiques dans une même méthodologie. Nous abordons également les mécanismes d'alternative et d'optionnalité permettant d'adapter la méthode au contexte de développement.

4.1.2.1 Processus et modèles GL

Concernant la conception des aspects fonctionnels (et métier) de l'application, nous conservons l'essentiel du processus Symphony originel, son approche conceptuelle puis organisationnelle des spécifications du système, ainsi que les modèles UML utilisés (c.f chapitre 3).

4.1.2.2 Processus et modèles IHM

Du point de vue de l'IHM, nous reprenons librement la méthode décrite par P. Renevier [95] et affinée par R. Chalon [20] (c.f section 1.3.2.2), pour la conception de système de réalité mixte collaboratifs, dont nous reprenons l'aspect « systèmes de réalité mixte », au détriment de l'aspect « collaboratifs ». Rappelons que cette méthode s'appuie essentiellement sur la définition de scenarii (abstraits puis concrets) afin de constituer un cadre de spécification de l'interaction homme-machine.

Nous intégrons au cadre de conception fournit par la méthode de P. Renevier les pratiques de conception suivantes :

- la réalisation d'un « dossier des prescriptions ergonomiques », point de départ du développement de l'interface homme-machine, dès la description des processus métier. Il s'agit alors pour l'ergonome responsable de la rédaction de ce dossier d'identifier, parmi les pratiques actuelles du métier, les goulots d'étranglement en termes d'efficacité opératoire, de redondance des tâches utilisateur, de communication entre acteurs externes ou internes etc. Ces différents points constitueront autant de valorisations possibles pour l'organisation ;

- une démarche de conception de l'IHM intégrant les principes de la conception centrée sur l'usage [29]. Ces principes sont bien évidemment appliqués dès la prise en considération de l'implication des utilisateurs finaux, c'est-à-dire les acteurs internes, dans le système développé ;
- l'intégration de la conception d'interfaces classiques, comme une alternative à la conception des interfaces de réalité mixte (voir ci-dessous).

Concernant les modèles issus de l'IHM, nous avons décidé d'utiliser une palette de modèles-clés issus du domaine : les arbres de tâches [93] et les diagrammes d'interaction spécifiques aux systèmes mixtes comme ASUR [46] et IRVO [21]. On notera que la méthode n'est pas limitative du point de vue des modèles utilisés, tant que les objectifs de l'activité dans laquelle ils s'intègrent ainsi que la cohérence intermodèles sont respectés.

4.1.3 Intégration des méthodologies GL et IHM

Si les activités en amont de notre méthode Symphony étendue ont pour objectif de permettre aux différents acteurs de la méthode d'acquérir une vision consensuelle du système à développer, d'autres activités se concentreront exclusivement sur ses aspects GL ou IHM. Ces activités n'impliqueront que les acteurs concernés par le domaine considéré. Il s'agit en effet de laisser toute latitude à ces derniers d'exprimer leurs compétences spécifiques. Nous proposons d'intégrer le processus de spécification de l'IHM en parallèle à celui de la spécification du noyau fonctionnel.

La méthode Symphony originelle propose une distinction au niveau de la spécification des besoins entre le découpage conceptuel des processus métier du futur système (phase de spécification conceptuelle des besoins) et l'organisation des concepts métier et des acteurs internes à ces processus métier (phase de spécification organisationnelle des besoins). Ces derniers constituent les utilisateurs finaux du système. Ils sont identifiés dès la phase d'étude préalable, avec l'objectif de déterminer leur implication dans les processus métier existants, toutefois leur implication dans la nouvelle version du système n'est détaillée qu'au cours de la spécification organisationnelle des besoins. La spécification de l'interface homme-machine ne peut donc intervenir qu'à partir de la phase de spécification organisationnelle.

Nous considérons qu'à l'issue de la phase spécification conceptuelle des besoins, la vision du métier, inscrite dans le découpage des processus métier, est suffisamment claire pour permettre à l'ergonome d'identifier les problèmes d'utilisabilité dans la réalisation courante des processus métier. À la suite de cette activité, le spécialiste IHM se joint à l'ergonome pour déterminer les types d'interaction (de réalité mixte ou classique) adaptés à chaque élément du découpage des processus métier. Ces deux rôles fonctionnels donc impliqué dans le cycle de développement avant que les acteurs internes (utilisateurs finaux) soient formellement assignés à l'exécution du processus métier, dans une optique d'étape préparatoire à la spécification de l'IHM.

Par ailleurs, la phase d'analyse décrit la mise en œuvre logicielle des spécifications du système, en traitant d'une part l'organisation des objets métier sous forme de classes et d'autre part la dynamique des interactions entre ces classes. Afin d'intégrer les éléments de l'IHM dans le futur système, nous décrivons également ces derniers sous forme de classes et d'interactions entre classes, avant de mettre en relation les entités GL et IHM.

Par conséquent, il est indispensable de s'assurer que les spécifications métier et interac-

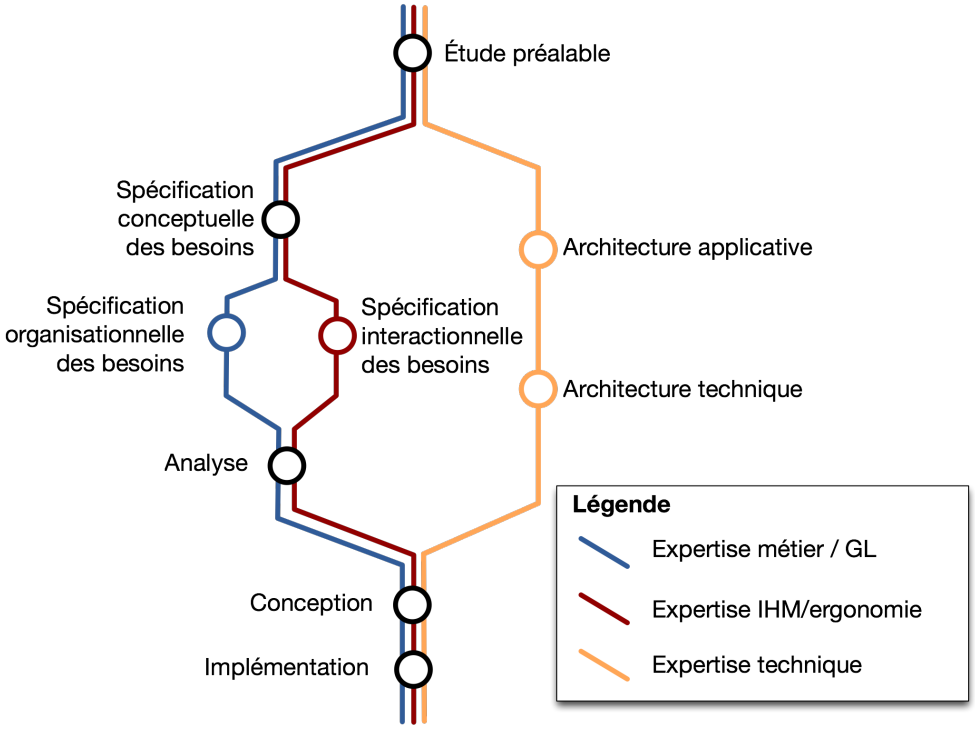


Figure 4.1 – Implication des rôles fonctionnels dans le cycle Symphony étendu

tionnelle proposent *in fine* une solution cohérente et unifiée aux besoins utilisateur. Nous présentons dans la suite de ce mémoire les activités collaboratives et les modèles consensuels permettant d'unifier les spécifications fonctionnelle et interactionnelle.

La figure 4.1 reprend les phases essentielles de la méthode Symphony étendue, ainsi que les différentes implications des rôles fonctionnels (auxquels nous avons intégré, à titre informatif, l'expertise technique correspondant à la branche droite de Symphony). Les mécanismes méthodologiques permettant soit de diviser le processus de développement, soit de réunir les fragments de processus, sont détaillés dans les sections suivantes.

4.1.4 Une méthode ouverte

Afin de permettre le choix entre plusieurs types de conception, notre méthode propose des mécanismes d'extension, en particulier un mécanisme d'alternatives. Celui-ci permet par exemple, en fonction des choix de conception effectués préalablement, d'envisager la conception d'un système interactif classique, ou d'un système de réalité mixte. Ces deux types de systèmes ont des contraintes et des besoins différents en termes de conception. La méthode prend ces aspects en considération en proposant des processus et modèles adaptés à chacun des objectifs, tout en respectant la cohérence générale de la démarche. Par exemple, seules certaines activités supplémentaires sont rajoutées ou adaptées dans le cadre de la conception d'un système de réalité mixte, telles que « choix des dispositifs d'interaction ».

Par ailleurs, la méthode n'est pas prescriptive : un nombre conséquent d'activités et de modèles ne sont préconisés que dans le contexte d'un développement portant sur un do-

maine métier mal maîtrisé par l'équipe de développement, ou bien d'interactions complexes nécessitant un développement méthodique.

4.2 Des activités collaboratives de conception

Dans le cadre d'une méthode de développement, les collaborations ne sont pas forcément médiées par la machine, contrairement aux travaux sur les collecticiels. Elles sont donc potentiellement beaucoup plus larges et les collecticiels ne peuvent être qu'un moyen de résoudre certains problèmes pratiques de collaborations. Les taxonomies [99, 39] proposées pour les collecticiels ne correspondent pas à nos besoins puisque les différentes collaborations sont décrites du point de vue de la machine.

Nous proposons donc plutôt de reprendre l'une des collaborations proposées par Grebici et al. [55]. Parmi les types d'activités collaboratives décrites par les auteurs, certaines sont centrées sur les processus, par exemple la coordination, consistant en une synchronisation des points de vue sur des modèles distincts, et d'autres sont centrées sur les modèles, dont la coopération, correspondant à la réalisation d'un modèle commun à plusieurs acteurs du développement.

Or, toutes les activités de notre méthode ont pour vocation de produire un artefact résultat (c.f section 4.4.2), utilisés lorsque nécessaire comme supports aux activités de collaborations entre acteurs. Par conséquent, nous n'utilisons par la suite que des activités de coopération, et assimilerons les termes de « collaboration » et « coopération ».

4.3 Un modèle pour définir la structure et l'organisation de l'interaction : les Objets Interactionnels

Selon la même approche et le même formalisme que les Objets Métier, nous proposons de représenter les processus et concepts élaborés au cours de la spécification de l'interaction sous forme d'*Objets Interactionnels* (OI).

4.3.1 Modélisation des Objets Interactionnels dans la phase de spécification organisationnelle et interactionnelle

Par symétrie avec l'espace métier, l'espace interactionnel est constitué d'Objets Interactionnels Processus, d'Objets Interactionnels Entité et d'Objets Interactionnels Données de référence :

- Les OI « **Processus** » prennent en charge la mise en œuvre des règles applicatives de gestion des Objets Interactionnels Entité et de construction de l'espace d'interaction, indépendamment des règles métier (par exemple, le processus de gestion d'une scène tridimensionnelle de réalité mixte),
- Les OI « **Entité** » décrivent les concepts centraux de l'espace interactionnel, ayant une forte densité sémantique ; ils représentent des concepts ayant un cycle de vie complexe, proposant des services ayant une forte cohésion ainsi qu'une forte cohérence avec la représentation du concept (par exemple, des marqueurs représentant l'utilisateur ainsi

que ses attributs – photographie, nom et prénom etc. –) sur la carte d'une application de géolocalisation),
- Les OI « **Données de référence** » représentant les données de base manipulées au sein de l'espace interactionnel (par exemple, des codes de couleur).

Nous illustrons le concept d'Objet Interactionnel en reprenant l'exemple de l'état des lieux. Par souci de concision (l'exemple complet étant déroulé au chapitre suivant), supposons que les activités de spécification de l'interaction proposent une interface homme-machine en réalité mixte, dont la cartographie serait composée des Objets Interactionnels suivants (représentés en figure 4.2) :

- un Objet Interactionnel Entité « Marqueur » (*Marker*), correspondant à la représentation d'une cible virtuelle ;
- un Objet Interactionnel Processus « Gérer scène 3D » (*Manage3DScene*) chargé de gérer les règles de construction d'une scène tridimensionnelle, dans laquelle seront positionnés des marqueurs ; cette dépendance entre Objet Interactionnel Processus et Objet Interactionnel Entité est matérialisée par une relation d'utilisation (*use*).

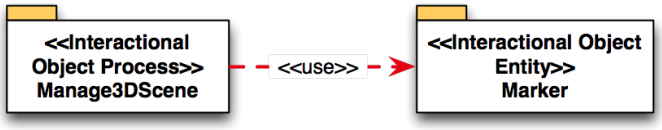


Figure 4.2 – Extrait de cartographie des Objets Interactionnels

Notons que l'identification des Objets Interactionnels d'un système soulève des questions comparables à celles posées pour l'identification des Objets Métier (voir la section 3.2). Toutefois, *a contrario* de la problématique métier de la réutilisation, encore épineuse à ce jour, le domaine de l'IHM a depuis toujours fortement factorisé ses composants – c'est-à-dire, ses interacteurs – dans des bibliothèques, frameworks et standards. À titre d'exemple, le standard HTML [1] définit exhaustivement les interacteurs mis à la disposition des développeurs.

Il ne s'agit évidemment pas de répliquer ou d'encapsuler ces approches déjà éprouvées. Que modélisent donc les Objets Interactionnels ? Inversement, comment les interacteurs plus simples sont-ils intégrés dans ce paradigme ? Afin de répondre à ces questions, nous appliquons les heuristiques suivantes :

1. La construction de l'espace interactionnel (scène virtuelle, espace de dialogue, etc.) est du ressort de l'OI Processus,
2. Un concept de l'espace interactionnel possédant plusieurs attributs caractérisant ses propriétés comportementales et sa représentation (c'est-à-dire, le rendu visuel et/ou audio) pourra être modélisé sous forme d'un OI Entité,
3. Un concept ne répondant pas au critère ci-dessus pourra être intégré à l'OI Processus comme partie de l'espace interactionnel global.

À la lumière de ces heuristiques, nous constatons que les représentations sous forme de formulaires (c'est-à-dire, de champs textuels, de radio boutons, de listes déroulantes etc.) sont peu adaptées à la modélisation suivant le formalisme des Objets Interactionnels. En

1. HTML 4.01 Specification – `http://www.w3.org/TR/html401/`

effet, selon ces règles, un formulaire, comme toute représentation trop fragmentée des concepts métier, sera intégré à l'Objet Interactionnel Processus.

Nonobstant les règles présentées ici, il convient de noter que les solutions de structuration de l'espace interactionnel en Objets Interactionnels relèvent également de l'expérience du concepteur, en particulier de sa perception de la pertinence de chaque concept du domaine.

Afin de permettre une collaboration efficace entre spécialistes GL et IHM pour les activités d'analyse, de conception puis d'implémentation du système, nous proposons un dispositf permettant d'expliciter les liens conceptuels reliant Objets Métier et Objets Interactionnels. Il s'agit concrètement de compléter le formalisme des Objets Symphony en y ajoutant la relation de dépendance « Représente ». La figure 4.3 illustre une application de la relation « Représente » (*represent*), entre les Objets Métier et Interactionnels de la réalisation de l'état des lieux.

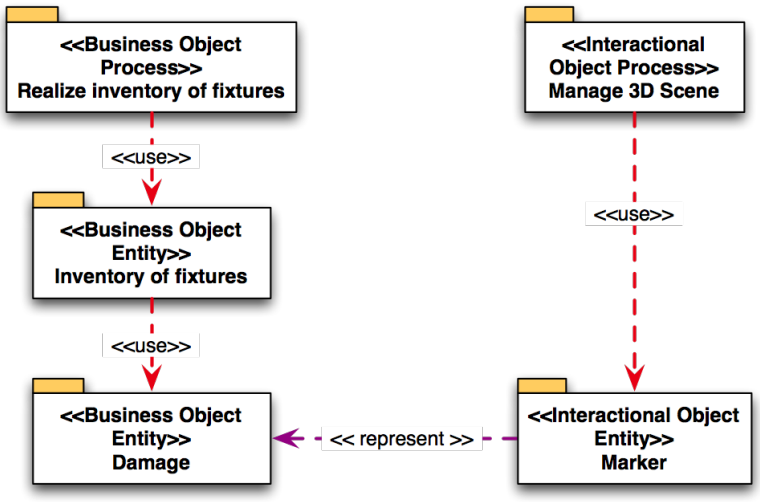


FIGURE 4.3 – Exemple d'utilisation de la relation « Représente »

4.3.2 Modélisation des Objets Interactionnels dans la phase d'analyse

À la suite des spécifications, les activités de la phase d'analyse guident le raffinement de la cartographie des Objets Interactionnels en un modèle de classes conceptuellement similaire à celui des Objets Métier (c.f figure 4.4).

Au cours de la phase d'analyse, les relations « représente » du modèle d'analyse lient les classes « Rôle » des Objets Interactionnels source aux classes « Interface » des Objets Métier cibles. Par exemple, dans la figure 4.5, on constate que les Objets Entité « Dommage » (*Damage*) et « Marqueur » (*Marker*) sont utilisés respectivement par l'Objet Métier Entité « État des lieux » (*Inventory of fixtures*, c.f figure 3.4, p. 94) et l'Objet Interactionnel Processus « GestionScène3D » (*Manage 3D Scene*, c.f figure 4.4), au travers des classes rôle « DommageLocalisé » (*Localized Damage*) et « MarqueurLocalisé » (*Localized Marker*). Ainsi, dans le modèle d'analyse, la relation de dépendence « représente » est tracée entre le rôle « MarqueurLocalisé » et l'Objet Métier Entité « État des lieux », qui détient le rôle « DommageLocalisé ».

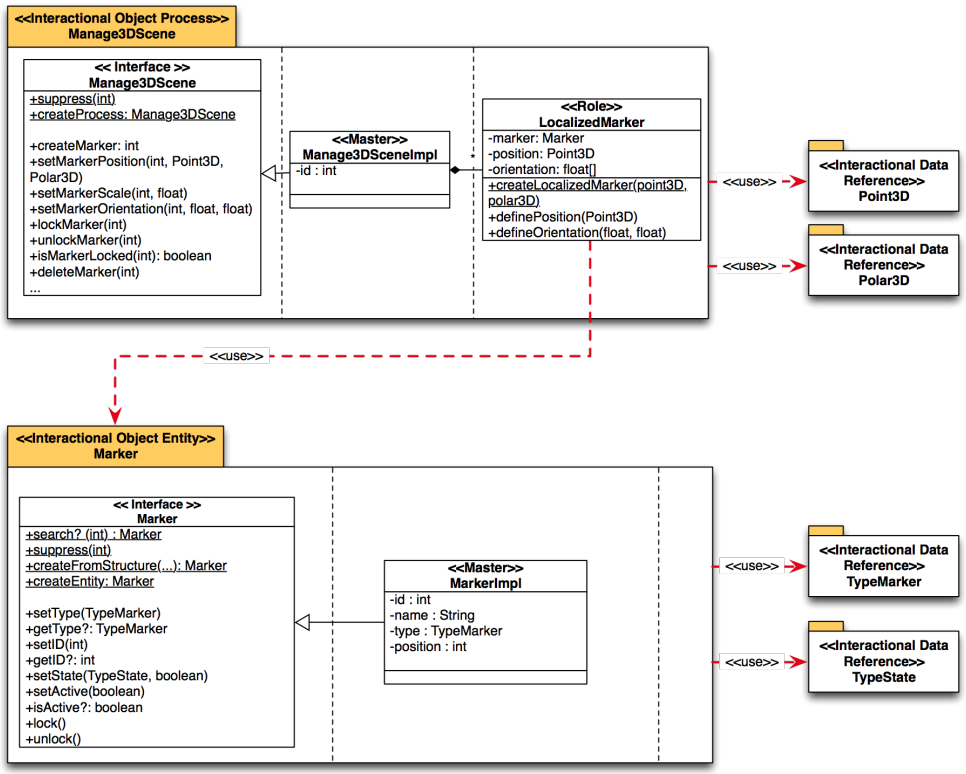


FIGURE 4.4 – Extrait du modèle d'analyse des Objets Interactionnels

Par la suite, il est nécessaire de décrire les mises en relation entre événements métier et événements interactionnels, ainsi que de traduire les concepts de l'interaction dans le référentiel du métier. Par exemple, le fait de déplacer les marqueurs de dommage dans la scène de réalité mixte peut modifier la relation du dommage à sa pièce, en fonction de la position du marqueur. Dans ce cas la position, qui est exprimée dans un référentiel virtuel, doit être traduite en une indication faisant sens du point de vue du métier, par exemple la pièce correspondante dans le plan d'architecte.

La description de la sémantique de la dynamique de la connexion entre Objets Interactionnels et Objets Métier peut être résumée par les activités suivantes :

1. L'identification des services des Objets Interactionnels pouvant avoir un impact sur l'espace métier, par exemple, le service « créerMarqueurLocalisé » (*createLocalizedMarker*). Nous utilisons des annotations UML pour identifier les méthodes concernées (voir figure 4.6), dans le modèle d'analyse des Objets Symphony. Cette approche de modélisation est détaillée au chapitre 7.
2. L'identification des services applicatif métier (c'est-à-dire, de l'Objet Métier Processus) pouvant être appelés en réaction à des événements d'interaction, par exemple, le service « createNewDamage » de l'Objet Métier Processus « Réaliser État des Lieux » (*RealizeInvOfFixtures*).
3. La description de la traduction de l'événement interactionnel en événement métier correspondant (par exemple, nous devons traduire les coordonnées en pixels du marqueur en une pièce du logement), et l'identification de l'événement de traduction

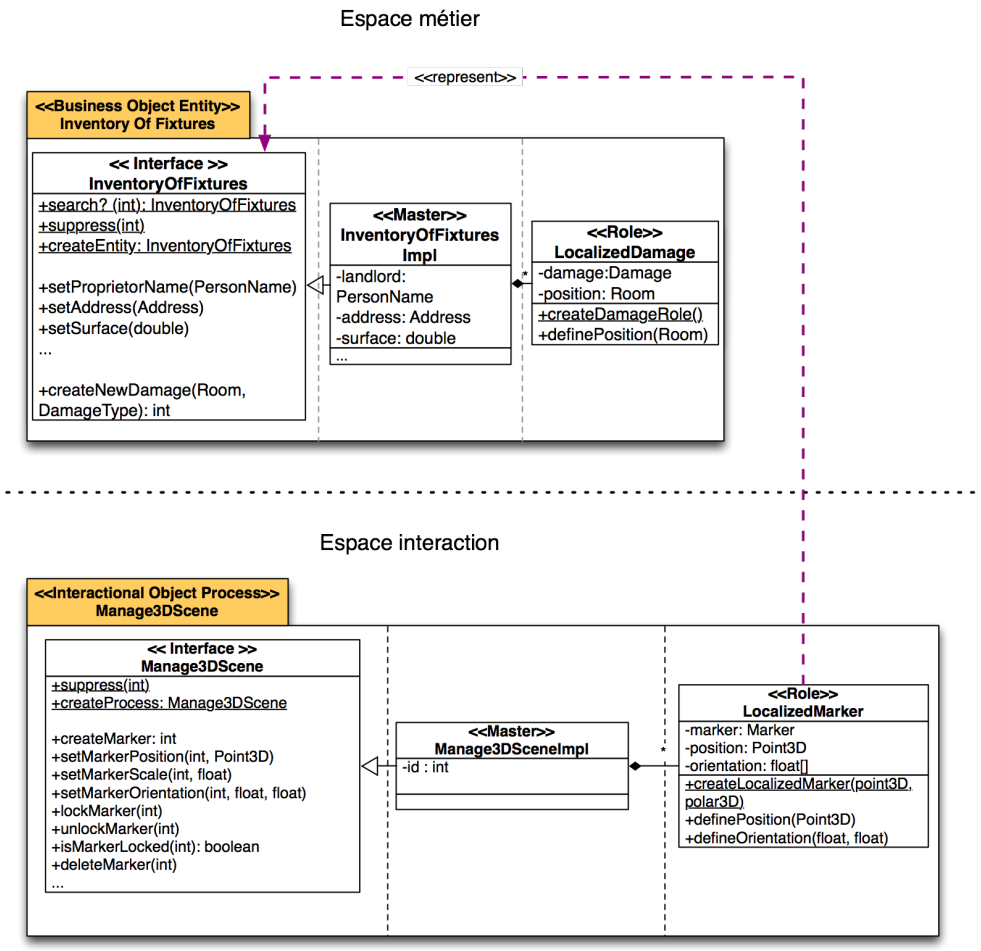


FIGURE 4.5 – Exemple de l'intégration de la relation « représente » au modèle d'analyse des Objets Symphony

(par exemple, l'événement « créerMarqueurDommage » (*createDamageMarker*) dans l'annotation de la figure 4.6).

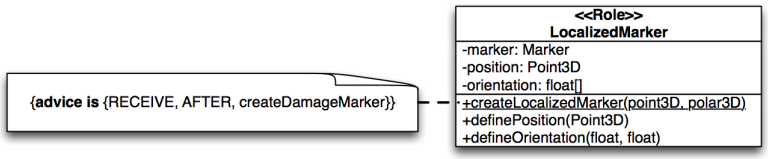


FIGURE 4.6 – Exemple d'annotation sur une classe « Rôle » issue de l'analyse statique de l'espace métier

Lors de cette activité, l'Objet Métier Entité cible de l'événement d'interaction n'est plus considéré directement. Afin de préserver les contraintes applicatives, les réactions aux événements d'interaction sont d'abord transmises à l'Objet Métier Processus (gérant ces dernières), auquel il incombe d'identifier l'Objet Métier Entité (ici, un objet « Dommage ») adéquat.

Des scenarii et diagrammes de séquences UML peuvent être utilisés pour formaliser cette

connexion Symphony, sous la forme d'une classe de contrôle nommée « Translation ». De par son rôle de médiateur entre fonctionnalités et interface, cette classe partage des caractéristiques avec la facette « Contrôle » d'un agent PAC [34] ou le « Contrôleur » d'un tryptique MVC [70].

La figure 4.7 présente un exemple de Translation, associée à la création d'un Objet Interactionnel Entité « Marqueur ». Comme mentionné plus haut, la réaction à l'événement à l'événement d'interaction consiste en un appel de méthode `createNewDamage`, réalisé par la Translation en direction de l'Objet Métier Processus. Il est en revanche possible d'effectuer des appels de méthode sur l'Objet Interactionnel Entité source de la Connexion Symphony, à condition que celui-ci ne modifie pas l'état de l'objet. Une Translation présente les propriétés suivantes :

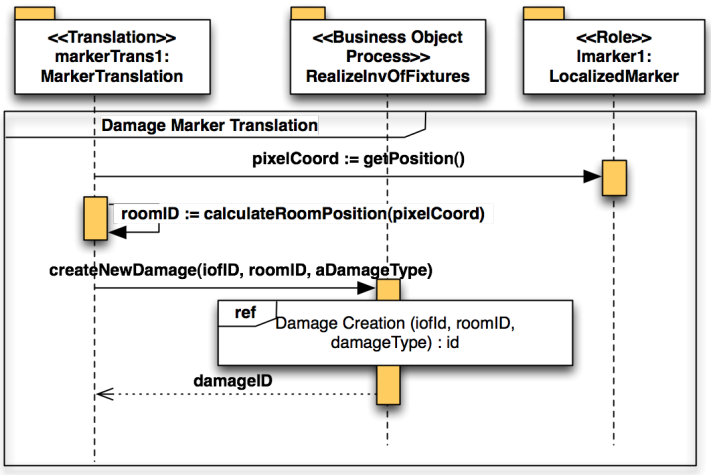


Figure 4.7 – Translation entre les rôles d'un marqueur et d'un dommage correspondant à l'événement « créerMarqueurDommage »

- Il existe au plus une seule instance de Translation pour chaque instance d'appariement entre rôles d'Objet(s) Interactionnel(s) et d'Objet Métier. Nous nommons le triplet :
$$c = \{[source_1, source_2, \ldots, source_n], cible, translation\}$$
une « Connexion Symphony ». Suivant notre exemple, la Connexion Symphony entre un marqueur localisé et un dommage localisé correspond au triplet suivant :
$$c1 = \{[marqueurloc_1], dommageloc_1, transMarqueur_1\},$$
- Chaque instance de Translation possède les références à sa ou ses sources (Objets Interactionnels) et sa cible (Objet Métier). Par exemple, l'instance de Translation $transMarqueur_1$ dispose des références aux objets $marqueurloc_1$ et $dommageloc_1$,
- Chaque traduction d'un événement interactionnel en événement métier est décrit à l'aide d'une méthode de la classe Translation. Un tel événement est appelé « événement de Connexion Symphony ». Dans notre exemple, l'événement interactionnel « createEntity » (appelé sur l'Objet Interactionnel « Marqueur », qui fournit un object $marqueurloc_1$) est traduit en une méthode « créerMarqueurDommage » appelée sur l'objet $transMarqueur_1$. Il est à noter que la traduction d'un événement de Connexion Symphony n'est pas systématiquement mis en relation avec un unique événement métier. Il peut au contraire nécessiter plusieurs appels (par exemple, pour obtenir la position de l'objet source, créer la cible et définir sa position),

- Si une instance de Translation donnée a, issue d'une Connexion Symphony c_1 obtient la référence d'un élément appartenant à une Connexion Symphony c_2, alors il est possible à l'instance de Translation a d'appeler les services de l'instance de Translation b associée à la Connexion Symphony c_2.

Grâce à ces quatre propriétés, la Translation constitue le seul point du modèle d'analyse (et, comme nous le verrons plus loin, du code d'implémentation) où concepts interactionnels et métier se rencontrent. De plus, ces propriétés sont suffisamment expressives pour couvrir tous les besoins des interactions IHM–noyau fonctionnel que nous ayons pu rencontrer jusqu'à présent.

4.4 Des modèles traçables et cohérents pour la collaboration et la communication

Les acteurs de la méthode génèrent, utilisent et manipulent, à travers les processus proposés, des informations structurées en produits de natures diverses (modèles, scenarii, dossiers). Nous regroupons ces éléments sous le terme générique d'artefacts.

Leur mise en œuvre réclame la prise en compte de deux problématiques orthogonales. D'une part, il est nécessaire d'assurer la traçabilité d'un raffinement d'artefact à l'autre et la cohérence des artefacts entre les différents aspects du système. D'autre part, deux types de modèles-clefs sont requis : les activités de coopération doivent être soutenues par des modèles consensuels pour tous les spécialistes impliqués dans leur élaboration et des modèles pour la communication avec les utilisateurs, décideurs etc. (ou *stakeholders*) doivent être établis. Enfin, ces deux principes peuvent être mis à profit pour estimer les produits du développement essentiels à la garantie d'un cycle de développement cohérent, centré sur l'usage et maintenable, et ainsi envisager la simplification de la méthode, en fonction des caractéristiques des projets développement.

4.4.1 Les modèles consensuels

Nous abordons dans cet intitulé les deux types de modèles consensuels utilisés dans la méthode Symphony étendue : les modèles collaborationnels et les modèles communicationnels, déjà abordés au chapitre 2. Les modèles collaborationnels font référence aux modèles utilisés au sein de l'équipe de développement pour permettre à des acteurs de métiers différents (par exemple, les spécialistes GL et IHM) de collaborer. Les modèles communicationnels, quant à eux, correspondent aux modèles utilisés par l'équipe de développement pour communiquer avec les utilisateurs et les décideurs.

4.4.1.1 Modèles collaborationnels : scenarii, diagrammes d'activités et Objets Symphony

La définition d'une démarche et la mise en œuvre de mécanismes de collaboration entre rôles fonctionnels ne suffisent pas pour permettre une intégration effective des méthodologies GL et IHM. En effet, il est indispensable de définir les modèles consensuels sur lesquels les experts des domaines métier et interactionnel pourront se baser, d'une part ; d'autre part, les mises en commun des fragments de démarche GL et IHM doivent être concrétisées par

des modèles exprimant le résultat de ces coopérations. Ce sont les modèles que nous avons décrits comme « collaborationnels » au chapitre 2.

De facto, nous recourrons à trois types de modèles collaborationnels : les **scenarii**, les **diagrammes d'activités UML** et les **Objets Symphony**.

Comme nous l'avons vu précédemment, les scenarii structurés au sens de Carroll [18] sont un moyen efficace de décrire les besoins utilisateurs à un niveau abstrait et amodal. De plus, à l'issue des travaux de Sutcliffe sur les convergences GL/IHM [114], ils ont été identifiés comme un point de jonction privilégié entre ces domaines. Les modèles raffinant les scenarii ne sont ainsi qu'une expression détaillée ou particulière, fonction d'un point de vue sur le système.

À l'instar des méthodes s'appuyant sur les principes de la conception centrée usage, nous utilisons les diagrammes d'activités UML pour représenter les tâches essentielles de l'utilisateur et leur organisation. Ceux-ci sont utilisés dans la méthode Symphony originelle pour raffiner les processus métier en activités, selon les objectifs suivants :

- la répartition des étapes de la réalisation du processus métier composant (c.f chapitre suivant) entre acteurs externes et acteurs internes ;
- la distinction entre activités manuelles (qui ne feront l'objet d'aucune informatisation) et activités informatisées (qui seront traitées par la phase d'analyse puis conçues et implémentées) ;
- la production, à l'issue d'une activité, d'un artefact métier utilisé soit par une activité de type différent (une activité manuelle produisant un artefact transmis à une activité informatisée, par exemple), soit par un acteur différent.

Les activités ainsi représentées sont structurellement et sémantiquement semblables à des tâches utilisateur essentielles, telles que décrites dans la méthode Wisdom [83] (voir chapitre 2). Les réponses à haut niveau d'abstraction du système ne sont pas représentées explicitement, mais sont détaillées par la suite au cours de la spécification interactionnelle des besoins.

Ces diagrammes d'activités UML, dont nous présentons un exemple en figure 4.8, constituent une base organisationnelle sur laquelle nous appuyons la réalisation des activités de spécification fonctionnelle et de spécification de l'IHM. De par leur formalisme simple et lisible, ils sont également adaptés aux besoins de la communication avec les utilisateurs et les décideurs.

Toutefois, les scenarii et les diagrammes d'activités ne sont pas des représentations satisfaisantes pour décrire le système, lorsque les besoins de modélisation se rapprochent du code. En raison des liens conceptuels et techniques appelés à exister entre l'espace métier et l'espace interactionnel, il est alors nécessaire d'adopter un formalisme structuré permettant d'identifier *in fine* les composants logiciels du système, leur organisation et les services qu'ils proposent, sans pour autant préciser les technologies utilisées (responsabilité du ressort de la branche technique de la méthode). Les Objets Symphony (c'est-à-dire, les Objets Métier et Objets Interactionnels) ont pour vocation de remplir ce rôle.

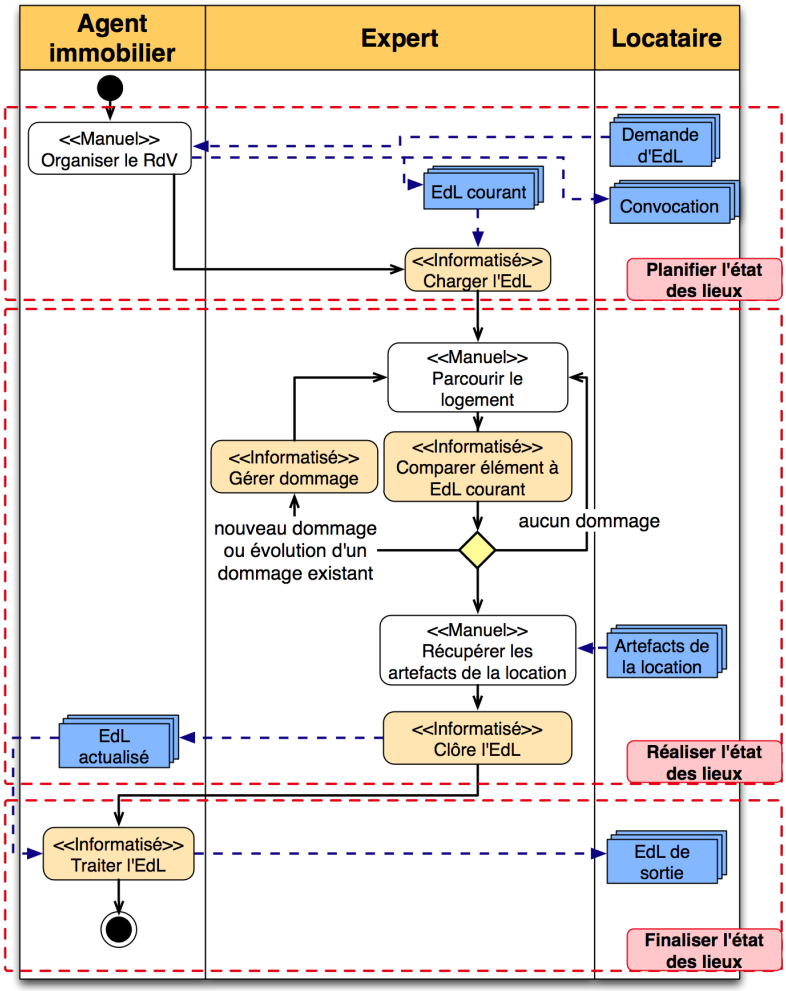


FIGURE 4.8 – Exemple de diagramme d'activités informatisées et manuelles, et répartition entre acteurs internes et externes, pour le processus métier « Gestion des états des lieux »

4.4.1.2 Modèles communicationnels : scenarii, diagrammes d'activités et prototypes

Rédigés en langue naturelle et orientés sur la problématique métier du système, les scenarii structurés sont également un outil de communication privilégié. Facilement compréhensibles par tous, ils servent ainsi de colonne vertébrale au développement.

De même, la réalisation du diagramme des activités informatiques et manuelles marque une étape essentielle dans le cycle de développement :

- le modèle décrit l'implication des acteurs internes dans la réalisation des processus métier composants ; il est dès lors possible de réaliser des évaluations d'utilisabilité centrées sur les utilisateurs ;
- le modèle précise les activités informatisées et manuelles, ébauchant ainsi la future organisation des processus métier.

Outre les données portées par le modèle, la forme de ce dernier, à la fois expressive et lisible, est adaptée à la communication avec utilisateurs et décideurs.

Cependant, les scenarii et diagrammes d'activités ne rendent pas compte de la réalité des solutions proposées. Les prototypes sont une approche largement employée pour répondre à ce besoin, à la fois pour permettre l'évaluation d'aspects du système (par exemple, les prototypes utilisés pour la validation de l'IHM, tel que décrit par l'ISO 13407, ainsi que la conception centrée usage) que pour la validation générale de l'application.

4.4.2 Les principes de cohérence et de traçabilité

Les artefacts de la méthode sont produits selon deux concepts fondamentaux :

- une logique de production/consommation ;
- la cohérence des relations sémantiques entre modèles.

La production/consommation d'artefacts permet de garantir une traçabilité des produits de la méthode, l'identification des processus métier jusqu'à l'implémentation. En effet, les relations de traçabilité étant transitives (si A permet de tracer B et B permet de tracer C, alors A permet de tracer C), tout produit peut être associé à un produit antérieur, dont il est le raffinement. De même, tout produit peut être associé à un processus métier, et donc à un incrément du projet de développement, facilitant ainsi la réutilisabilité des processus métier et leur maintenabilité.

Le second principe est essentiel pour la mise en œuvre des collaborations entre spécialistes : il s'agit en effet de définir des relations sémantiques entre des modèles complémentaires et de même formalisme, tels que les Objets Métier et Objets Interactionnels, et des relations sémantiques entre des modèles de différents niveaux de raffinement et de formalisme différent. Par exemple, ce principe nous permet de signaler que le diagramme des activités manuelles et informatisées (voir ci-dessus) fait référence aux processus métier composants identifiés plus en amont du cycle de développement. Dans le contexte de nos travaux, nous limitons ces relations sémantiques à des correspondances lexicales. Nous décrivons au chapitre 7 une opérationnalisation des règles de cohérence entre les produits de la méthode.

4.4.3 Profilage de la méthode Symphony étendue

La méthode Symphony étendue n'est pas prescriptive : la plupart des activités et des produits ne sont en effet utilisés que si la complexité du domaine métier ou l'expérience de l'équipe de développement le justifient. Cependant, au-delà d'un certain niveau de simplification du cycle de développement, nous estimons qu'il n'est plus envisageable de garantir une cohérence et un traçabilité satisfaisantes des artefacts produits, compromettant ainsi la maintenabilité du système développé. De plus, certains artefacts, tels les modèles communicationnels, sont essentiels à la validation des incréments du projet par les utilisateurs et les décideurs. De même, les modèles collaborationnels sont indispensables à la collaboration entre spécialistes métier et spécialistes de l'interaction homme-machine.

Considérant les modèles pour la collaboration et la communication d'une part, et les principes de cohérence et traçabilité d'autre part, nous proposons une structure minimale de produits du développement (c.f figure 4.9), basée sur les **scenarii**, les **diagrammes d'activités** et les **Objets Symphony**, les **prototypes**.

Il s'agit en effet du sous-ensemble de produits requis pour garantir trois propriétés :

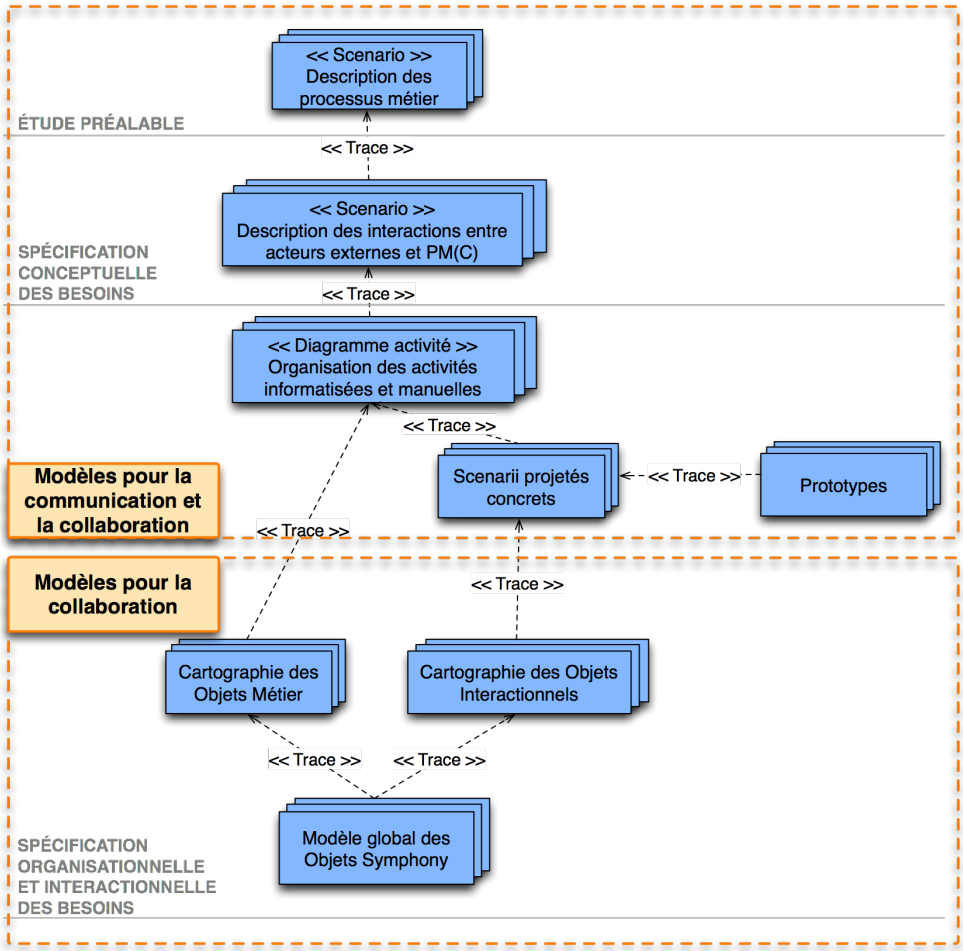


FIGURE 4.9 – Ensemble minimal de produits du développement

- la production des modèles communicationnels nécessaires pour permettre aux utilisateurs et décideurs de valider les étapes clefs du développement (formalisation des besoins, conceptualisation de l'interaction homme-machine) ;
- la production des modèles collaborationnels nécessaires pour permettre aux acteurs du GL et de l'IHM, d'une part, de définir une vue consensuelle du développement, notamment des processus métier et de l'organisation des activités informatisées et manuelles, et d'autre part d'intégrer les contributions élaborées lors des sous-processus parallèles de développement ;
- une traçabilité continue des produits, de l'étude préalable jusqu'à l'analyse.

Dans la section suivante, nous abordons l'outillage de la méthode Symphony étendue.

4.5 Un outillage omniprésent

Afin de favoriser la mise en œuvre efficace de la méthode, les processus, modèles, savoir-faire, etc., doivent être décrits de manière suffisamment précise, détaillée et structurée pour être utilisables par les intervenants de la méthode. Une fois ces éléments détaillés, il est nécessaire de fournir un ensemble d'outils facilitant l'exploration de la méthode, la recherche de

solutions, la planification de l'ensemble du développement de l'application, l'automatisation des activités systématiques de développement (le cas échéant), la génération de modèles et de code. Nous exposons dans cette section l'outillage destiné à la documentation du processus. L'instrumentation à l'exécution et l'instrumentation des modèles sont décrits aux chapitres 6 et 7.

4.5.1 Spécification et instrumentation du processus de la méthode

En ce qui concerne la description de la méthode, nous avons choisi de décrire nos propositions à l'aide de patrons, ainsi que proposé dans le cadre de la méthode Symphony étendue par Jausseran [63]. Nous avons également repris le formalisme P-Sigma, dont l'expressivité convient pour décrire les différentes activités et modèles de notre méthode.

Comme nous l'avons vu dans la section 2.3, la description d'un système de patrons pertinent ne suffit pas à la mise en œuvre d'une méthode efficace. Il est en effet indispensable de disposer d'un outil adapté pour faciliter la navigation parmi les patrons de notre méthode.

À l'instar de la méthode Symphony originelle, notre méthode est donc instrumentée par un outil de gestion de patrons : AGAP Lite, une refonte de l'outil AGAP décrit à la section 3.4.6, caractérisé par une meilleure souplesse et un meilleur guidage que l'outil original. La figure 4.10 présente une capture d'écran de ce nouvel outil.

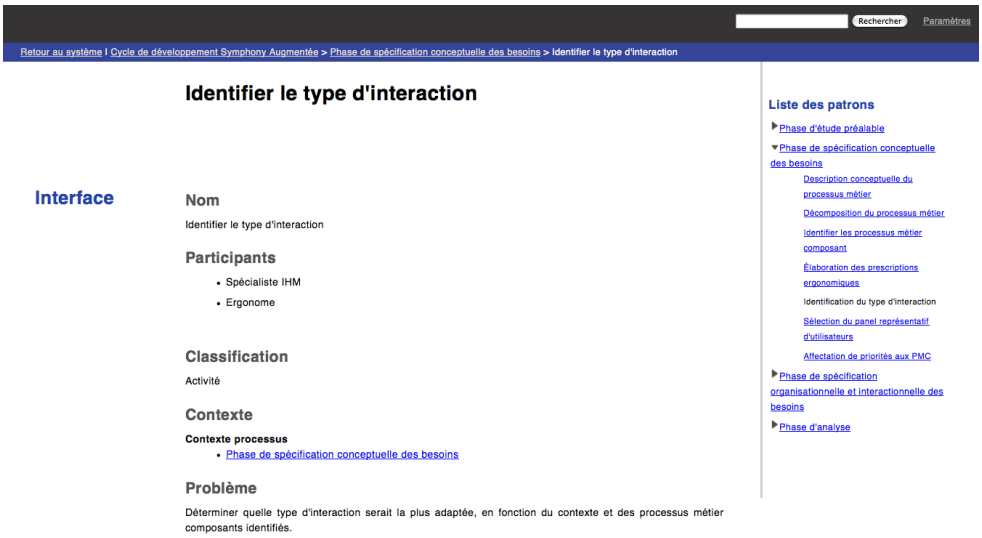


Figure 4.10 – Capture d'écran de l'outil de gestion de patrons AGAP Lite

En particulier, nous avons intégré la technique d'interaction dite du fil d'Ariane (*breadcrumb*, partie supérieure de la capture d'écran), afin de faciliter le positionnement de l'utilisateur dans le système de patrons.

De plus, nous avons modifié la barre latérale de navigation : celle-ci est désormais organisée par niveaux de hiérarchie déroulants (à droite de la capture d'écran). Afin de minimiser la quantité d'informations présentée à l'utilisateur, la liste des patrons est condensée, afin de ne montrer que les patrons de la phase courante (seuls les patrons de phase sont signalés).

Enfin, les relations entre patrons sont désormais intégrées de manière transparente :

- la relation « utilise » signale qu'un patron $P1$ utilise un patron $P2$; cette information apparaît déjà dans la solution formelle du patron courant, ou bien les patrons apparaissent comme des liens hypertextes dans la solution textuelle ;
- une relation « alternative » d'un patron $P1$ dénote qu'il existe un patron $P2$, traitant le même problème que $P1$, mais en privilégiant des forces différentes ; par conséquent, les alternatives pour chaque patron apparaissent adjacentes à la rubrique « Forces », et détaillent les forces du patron proposé en alternative ;
- une relation « raffine » d'un patron $P1$ implique qu'il existe un patron $P2$ abordant un problème plus spécifique que $P1$; les liens vers les patrons raffinant le patron courant apparaissent donc adjacents à la rubrique « problème », et détaillent le problème plus spécifique traité par le patron proposé ;
- une relation « requiert » d'un patron $P1$ précise que l'application d'un patron $P2$ est requise avant de pouvoir réaliser la solution de $P1$; ce type de relation apparaît soit comme une activité précédent le patron courant (dans la solution formelle), soit les produits du patron requis sont requis patron courant (et apparaissent également dans la solution formelle).

Quant à la souplesse d'utilisation, l'utilisateur a désormais la possibilité de modifier la configuration de plusieurs rubriques (par exemple, la liste et la description des acteurs du développement) à tout moment.

Grâce à ce guide méthodologique, les concepteurs vont pouvoir effectuer des recherches, naviguer en suivant les processus préconisés par la méthode, identifier les étapes franchies et restantes et envisager la suite des opérations de modélisation et de conception à mener pour aboutir à un système de réalité mixte opérationnel. À ce jour, 21 patrons ont été entrés dans AGAP Lite, couvrant les phases d'étude préalable jusqu'à l'analyse.

4.6 Synthèse

Nous avons montré dans cette section comment nous souhaitions mettre en œuvre les principes directeurs de notre méthode à savoir intégrer les habitudes de travail des concepteurs, garantir la traçabilité et la cohérence entre les modèles, définir des modèles consensuels pour la collaboration et la communication, préconiser des activités collaboratives de conception et fournir un outillage pour la conception. Ces principes nous ont amené à effectuer des choix définissant la structure et l'organisation de notre méthode :

- du point de vue des modèles, nous utilisons les modèles d'UML, les arbres de tâches et des diagrammes d'interaction de type ASUR. Ces choix ne sont toutefois pas limitatifs et peuvent être adaptés aux habitudes de développement, à l'expertise disponible, ainsi qu'au type d'interface développé (en réalité mixte ou classique)
- notre démarche se base sur la méthode Symphony originelle et sur les travaux de Renevier [95] pour la conception de l'interaction en réalité mixte. Elle est décrite dans des patrons sous forme de diagrammes d'activités UML commentés en langue naturelle,
- les Objets Symphony, les diagrammes d'activités UML et les scenarii sont utilisés comme modèles consensuels supportant les coopérations entre experts ; les scenarii et les prototypes sont utilisés comme modèles communicationnels,
- l'ensemble des produits de la méthode sont définis comme traçables et leur cohérence est vérifiée via l'instrumentation des modèles,

- notre démarche est instrumentée par un guide méthodologique fourni sous la forme d'un site web indépendant qui a été généré par l'atelier de gestion de patrons AGAP,
- les applications développées avec Symphony voient leur implémentation et leur exécution simplifiée grâce à un intergiciel.

*

* *

Dans la section suivante, nous démontrons l'application des différents principes décrits ici sur notre méthode Symphony étendue. Nous nous appuyons sur un cas d'étude : le développement d'une application destinée à faciliter la réalisation d'états des lieux.

CHAPITRE 5

Application de la méthode Symphony étendue

Un changement en prépare un autre

NICOLAS MACHIAVEL

NOUS APPLIQUONS dans ce chapitre la méthode Symphony étendue, construite à partir des principes décrits au chapitre précédent, sur un cas d'étude. Chaque phase de la méthode est introduite par un diagramme d'activités UML reprenant les activités la composant. Certaines activités particulièrement denses sont également décrites sous forme de diagramme d'activités. Pour d'évidentes raisons de concision, il ne nous est pas possible de détailler l'intégralité des choix et des activités de la méthode. Nous nous concentrons ici sur les aspects de notre méthode qui la distinguent des travaux plus classiques tels que RUP [72]. Signalons à ce titre que les activités de coopération entre acteurs du développement sont signalées par des labels « COOP ».

Enfin, l'ensemble de notre méthode est accessible en ligne [1]. Le lecteur intéressé y trouvera une description détaillée de chaque activité et produit, ainsi que de nombreux exemples, dont une version étendue du cas d'étude présenté dans ce mémoire.

En préalable au déroulement du cas d'étude, nous présentons dans la section suivante les évolutions principales différenciant la méthode Symphony originale de notre propre version.

5.1 Évolution de la méthode Symphony

Nous avons décrit dans le chapitre précédent les principes fondamentaux de notre méthode de développement des systèmes de réalité mixte. Ceux-ci sont intégrés à la méthode Symphony, décrite au chapitre 3.

1. http://iihm.imag.fr/godetg/PatternSystem/

La figure 5.1 présente l'évolution opérée entre la méthode Symphony originelle (c.f figure 5.1(a)) et notre version étendue (c.f figure 5.1(b)). Les phases affectées par cette évolution sont signalées en marquées d'un recouvrement. Un sous-processus itératif d'évolution des aspects métier, non représenté par souci de lisibilité, couvre également les phases d'étude préalable, de spécification conceptuelle des besoins et de spécification organisationnelle et interactionnelle des besoins. De plus, nous résumons dans les tableaux 5.1 à 5.3 les caractéristiques de la méthode Symphony étendue.

TABLEAU 5.1 – Grille d'analyse du processus Symphony étendue

	Phases couvertes	Phases couvertes par le développement de l'IHM	Phases intégrant des collaborations
Modélisation du métier	M		{M, G}, {M, G, I, E}
Spécification des besoins	M | G | I | E	I | E	{M, G}, {M, G, I, E}, {I, E}, {G, I}
Analyse	G | I	I	G, I
Conception	G | I	I	G, I

M : Spécialiste métier ; G : Spécialiste GL ; I : Spécialiste IHM ; E : Ergonome

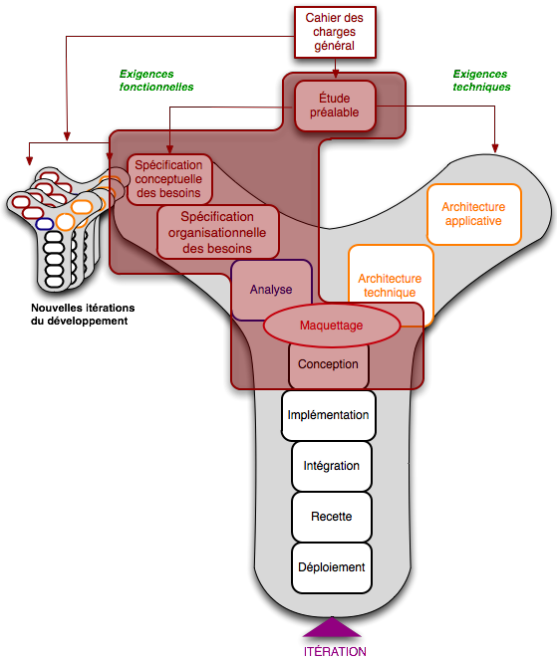
TABLEAU 5.2 – Grille d'analyse des modèles Symphony étendue

	Modélisation d'aspects métier/GL	Modélisation d'aspects IHM/ergonomie	Modèle de mise en correspondance
Modèle GL	UML, Objets Métier	Objets Interactionnels	Objets Symphony {Collab}, Diagrammes d'activités {Comm, Collab}
Modèle IHM		CTT, ASUR	
Scénario	(Cas d'utilisation {Comm, Collab}), Processus Métier {Comm, Collab}	Scenarii abstraits et concrets {Comm}	
Prototype	{Comm, Collab}		

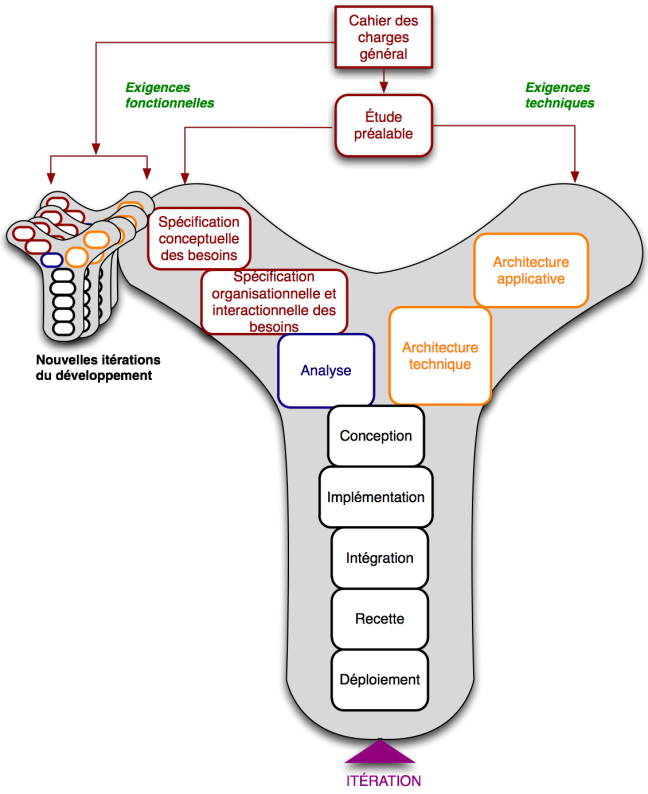
Collab : Modèles collaborationnels ; Comm : Modèles communicationnels

TABLEAU 5.3 – Grille d'analyse de l'instrumentation de Symphony étendue

Documentation de la méthode	Instrumentation du processus	Instrumentation à l'exécution
Outil de gestion de patrons AGAP Lite	• Outils d'aide à la conception UML • Transformations de modèles et vérifications de cohérence	Intergiciel Sonata



(a) Modifications opérées sur le cycle de développement originel



(b) Nouveau cycle de développement

Figure 5.1 – Évolution de la méthode Symphony

5.2 Présentation du cas d'étude

Nous prenons comme exemple illustratif l'informatisation d'une agence immobilière. Le domaine métier de la gestion de biens immobiliers consiste généralement en un traitement des logements à la charge de l'agence, tel que les locataires les occupant, leurs propriétaires, le suivi des charges locatives, des réparations, des réunions de copropriété.

Nous nous concentrons sur un problème récurrent rencontré par les employés d'agence immobilière chargés de réaliser les états des lieux : l'incomplétude des informations disponibles pour évaluer l'état d'un logement. En effet, la plupart des processus métier d'agence immobilière liés aux états des lieux sont informatisés sommairement. En particulier, les détails concernant la nature et la disposition des dommages sont la plupart du temps décrits par des formulaires papier.

Ces données sont souvent insuffisantes lorsque l'agent doit rendre compte de l'évolution d'un dommage ou d'une usure particulière d'une occupation à l'autre. Ce manque devient problématique lors des contentieux entre propriétaire, expert, locataires et agence immobilière.

Nous décrivons dans les sections suivantes le résultat de l'application de la branche fonctionnelle de la méthode Symphony étendue sur ce cas d'étude. La phase de conception sera abordée dans la prochaine partie (c.f chapitres 6 et 7).

Cette mise en œuvre de la méthode correspond à un développement logiciel intégrant un fort risque technologique. Notre méthode est particulièrement adaptée à ce type de projet, pour lequel il est recommandé de construire une documentation précise des itérations de conception, notamment lorsque l'équipe de développement n'est pas familière avec le contexte métier et/ou les technologies. Un projet plus « simple » ou bien une équipe constituée d'experts des systèmes de réalité augmentée ne nécessiterait donc probablement pas autant de documentation.

5.3 Étude préalable du métier

Une étude préalable du métier est réalisée avant même la première itération du développement (c.f figure 5.2). L'objectif général de la phase est d'obtenir une décomposition fonctionnelle du métier, telle que pratiquée par l'organisation cliente, afin d'identifier les processus métier et leurs participants. La plupart de ces activités sont réalisées en coopération afin de favoriser, pour tous les acteurs du développement, l'acquisition d'une vue unifiée du domaine métier et de sa décomposition en processus métier. Nous reprenons ici la définition d'un processus métier (PM) comme une séquence d'activités internes réalisée dans le domaine métier étudié, dont le but est d'obtenir un résultat observable et dont la valeur apportée à un utilisateur individuel peut être mesurée.

Le **recueil des besoins** consiste, pour l'essentiel, à restructurer les données du cahier des charges, avec pour objectif d'identifier clairement les besoins fonctionnels et non-fonctionnels, le périmètre du projet, ainsi que le domaine métier traité.

Il s'agit ensuite d'opérer un **découpage fonctionnel du métier**, permettant d'identifier les processus métier essentiels de l'organisation, ainsi que les acteurs externes impliqués dans la réalisation de chaque processus métier. Processus métier et acteurs externes sont décrits

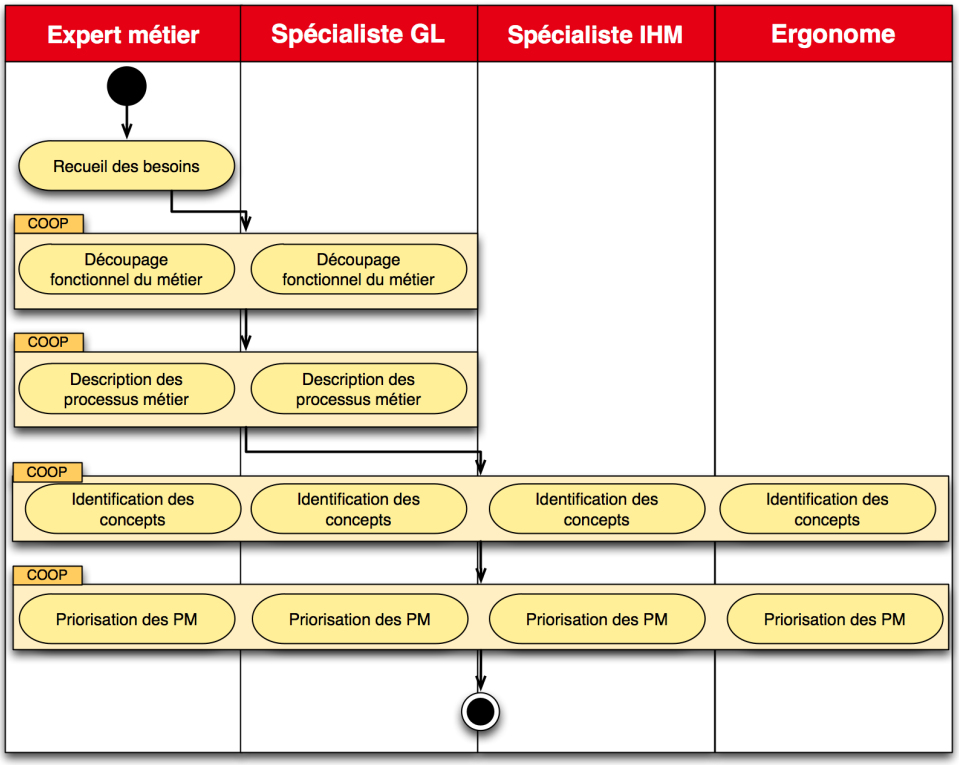


Figure 5.2 – Phase d'étude préalable

respectivement sous la forme de cas d'utilisation et d'acteurs UML.

Une **description des processus métier**, c'est-à-dire un scénario structuré faisant ressortir les acteurs internes et externes, les concepts principaux du processus métier, ainsi que les événements centraux du processus, est ensuite effectuée. L'encart présenté en page 134 éclaircit les notions d'acteur externe, interne, principal ou secondaire. Nous reprenons dans le tableau 5.4 la description du processus métier « Gestion des états des lieux », déjà présentée au chapitre 3.

Une fois les processus métier décrits, l'**identification des concepts** vise la rédaction d'un glossaire des concepts identifiés au cours du découpage fonctionnel du métier. Ce glossaire constitue un référentiel commun de compréhension et de communication pour l'équipe de développement.

Enfin, les processus métier sont priorisés (activité de **priorisation des processus métier**), avec l'objectif de réaliser dès la première itération un incrément à forte valeur ajoutée. Par souci de concision, nous assumons que le processus métier « Gestion des états des lieux » répond à ce critère. La figure 5.3 résume les activités de découpage du métier, d'identification des acteurs externes, ainsi que la priorisation du développement des processus métier, dans le contexte du développement de l'application de gestion des états des lieux.

TABLEAU 5.4 – Description du processus métier « Gestion d'un état des lieux »

- L'**agent immobilier (acteur interne)** *organise un rendez-vous (événement)* entre le **locataire (acteur externe)** (sortant dans le cas d'un état des lieux de sortie, entrant dans le cas d'un état des lieux d'entrée) et l'**expert état des lieux (acteur interne)** (salarié de l'agence immobilière, huissier, sous-traitant, etc.), choisi par l'agence immobilière chargée de réaliser l'***état des lieux (ressource)*** ;
- l'**agent immobilier** *fournit* à l'**expert état des lieux (EdL)** les informations nécessaires à la bonne réalisation de l'***EdL*** (telles des fiches de renseignements à remplir, les états des lieux précédents, si nécessaire, la liste des artefacts à récupérer etc.) ;
- l'**expert EdL** *réalise* l'***EdL***, en présence du **locataire**, qui peut émettre des objections quant aux constats de l'**expert EdL** ;
- en cas de désaccord sur le contenu de l'***EdL***, une *comparaison est faite* par l'**expert EdL**, avec l'***EdL précédent*** si celui-ci est disponible ;
- les artefacts du logement (clefs diverses, par exemple) sont soit *récupérés* (EdL sortant), soit *remis* (EdL entrant) au **locataire**. L'***EdL*** est signé par l'**expert EdL** et le **locataire** (sauf en cas de désaccord majeur) ;
- l'**expert EdL** *transmet l'état des lieux* à l'**agent immobilier**, qui *met le dossier du logement et de location à jour* ;
- si nécessaire, l'**agent immobilier** contacte le **propriétaire (acteur externe)** pour discuter d'éventuelles réparations à apporter, ou simplement des usures ou dégradations constatées, et envisager une remise à niveau du logement.

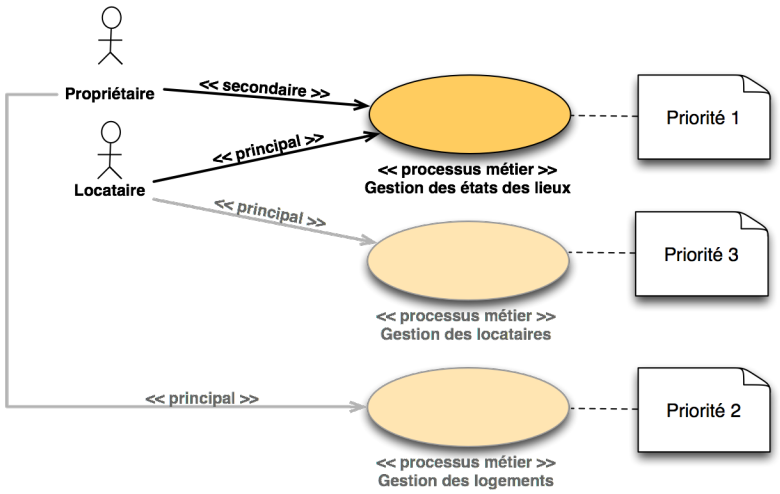


FIGURE 5.3 – Découpage fonctionnel du métier, priorisation du développement et identification des acteurs externes (exemple de l'agence immobilière)

5.4 Spécification conceptuelle des besoins

Cette phase, décrite en figure 5.4, détaille un processus métier choisi parmi ceux identifiés au cours de l'étude préalable. Son objectif est de guider la description de ce processus métier, l'identification de ses objectifs, des contraintes éventuelles, ainsi que de la valeur ajoutée pour chaque acteur externe.

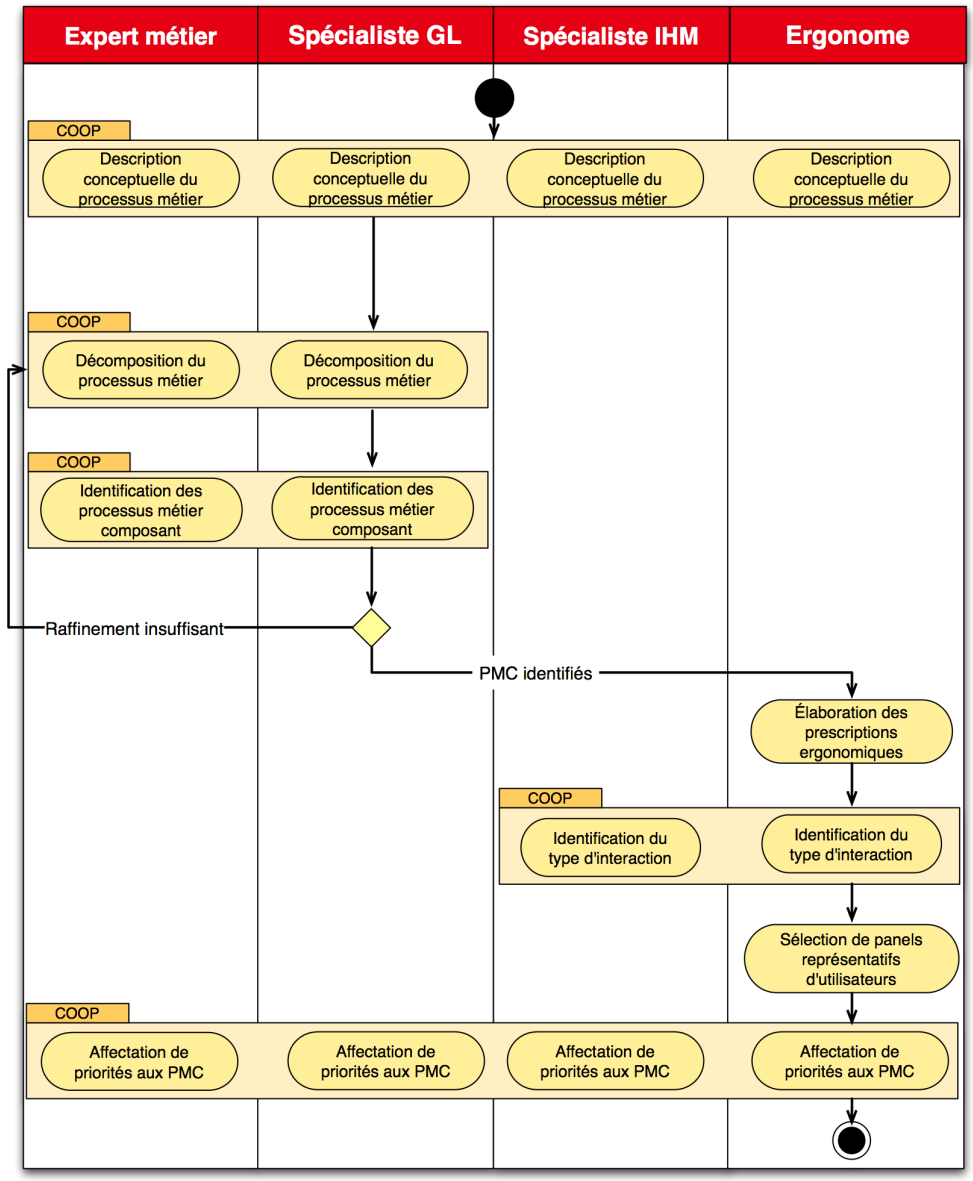


Figure 5.4 – Phase de spécification conceptuelle des besoins

L'activité de **description conceptuelle du processus métier** consiste à décrire le processus métier sous la forme d'un diagramme de séquences UML impliquant les acteurs externes et le processus métier, et éventuellement de scenarii structurés mettant en contexte le processus métier. Les scenarii permettent ainsi aux acteurs du développement, ainsi qu'aux utilisateurs et décideurs, d'acquérir la même vision initiale de l'organisation de l'entreprise.

Acteurs externes, acteurs internes

Les acteurs impliqués dans le système peuvent être classifiés selon deux catégories orthogonales :

1. Acteur **principal** (directement impliqué dans l'utilisation du système, par exemple en tant que client d'un service) ou acteur **secondaire** (impliqué dans la réalisation d'un service, sans être bénéficiaire du résultat) ;
2. Acteur **interne** (réalisant physiquement les actions sur le système, à la manière d'un opérateur) ou acteur **externe** (ne participant pas à la réalisation physique des actions sur le système).

Parmi les combinaisons possibles, on doit identifier pour le système **au moins un acteur principal externe**, c'est-à-dire qu'un acteur doit forcément bénéficier du résultat de l'application. On notera que dans le cas d'événements déclenchés par des contraintes d'ordre temporel, on pourra assimiler l'acteur externe à une "horloge système".
La classification des acteurs peut se déduire des réponses aux questions suivantes :

Type de question	Acteur correspondant
Qui bénéficie de l'utilisation du système ?	Principal
Qui interagit avec le système sans que le bénéfice final du processus métier lui soit destiné ?	Secondaire
Qui utilise le processus métier ?	Interne
Qui est déclencheur du processus métier ?	Externe

Exemple de la pompe à essence
En fonction des processus métier, une même personne physique peut endosser différents rôles. Par exemple, un usager voulant utiliser une pompe à essence est à la fois un acteur principal externe (il décide d'avoir recours aux services d'une pompe à essence) et acteur principal interne (il se sert lui-même) au cours d'un même processus métier.
En revanche, un usager en déplacement aux états-unis ne sera parfois qu'acteur principal externe : il suffit d'entrer dans certaines stations services, et c'est un pompiste qui sert le client. Dans ce cas, le pompiste est un acteur interne (c'est lui qui utilise le système, c'est-à-dire la pompe à essence), mais secondaire : en effet, il ne perçoit pas de bénéfice direct du processus, dont le but premier est de remplir le réservoir à essence de la voiture.

Suivant cette description, chaque PM est décomposé en sous-processus (activité de **décomposition du processus métier**) ininterruptibles (en considérant un déroulement nominal du processus métier), identifiés sous le nom de Processus Métier Composants (PMC) (activité d'**identification des processus métier composant**). Suivant notre cas d'étude, le processus métier « Gestion des états des lieux » est composé de trois échanges ininterruptibles : « Planifier l'état des lieux », « Réaliser l'état des lieux », et « Finaliser l'état des lieux » (c.f figure 5.5).

Chaque PMC est enfin lui-même décrit, en prenant en compte les cas alternatifs au scénario nominal (conditions, cas d'erreur ou d'exception).

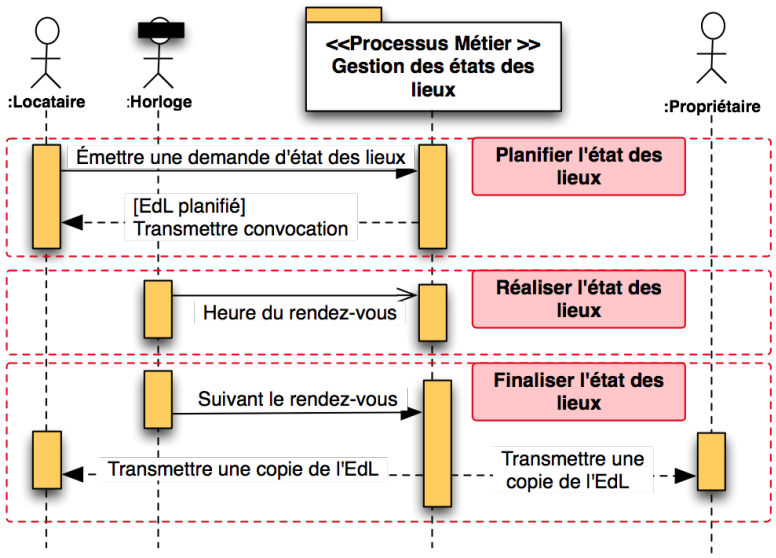


FIGURE 5.5 – Raffinement du Processus Métier « Gestion des états des lieux »

Une fois les processus métier composant identifiés, l'ergonome évalue les processus opératoires du métier existants, afin de relever les problèmes d'utilisabilité des systèmes informatiques existants, les goulots d'étranglement des échanges informatisés ou non. Une liste de prescriptions ergonomiques répertorie ces éléments (activité d'**élaboration des prescriptions ergonomiques**), à prendre en compte pour le système en cours de développement.

Nous assumons que les ergonomes suggèrent de privilégier deux critères d'ergonomie au sens de Bastien et Scapin [7] : la charge de travail minimale et le guidage. La charge de travail minimale est définie comme « l'ensemble des éléments de l'interface qui ont un rôle dans la réduction de la charge perceptive ou mnésique des utilisateurs et dans l'augmentation de l'efficacité du dialogue » et le guidage est décrit comme « l'ensemble des moyens mis en œuvre pour conseiller, orienter, informer, et conduire l'utilisateur lors de ses interactions avec l'ordinateur (messages, alarmes, labels, etc.), y compris dans ses aspects lexicaux ».

Nous envisageons le développement d'une interface intuitive, minimaliste du point de vue des données présentées à l'utilisateur, tout en permettant à ce dernier d'accéder aux différentes fonctionnalités du système en un nombre d'actions réduit.

Lors de l'activité d'**identification du type d'interaction**, le spécialiste IHM et l'ergonome collaborent afin de déterminer les types d'interaction pouvant être envisagés pour l'application, telles que les interfaces de réalité mixte ou les interfaces classiques. Bien que les acteurs internes, identifiés au cours de l'étude préalable, ne soient pas formellement assignés à la réalisation des processus métier, ils sont néanmoins pris en compte dans la réalisation de cette activité. Dans le cas de notre exemple, le PM « Réalisation d'un état des lieux » est un bon candidat pour une interaction en réalité mixte, pour les raisons suivantes :

- Le cas d'étude décrit une situation où l'utilisateur (l'expert commissionné par l'agence immobilière, acteur interne) ne peut pas utiliser de station de travail fixe lors de la réalisation de l'activité,
- Les descriptions textuelles sont à la foix imprécises et laborieuses à rédiger et utiliser, en particulier lorsqu'il s'agit de décrit l'évolution d'un dommage au cours du temps,

- Plusieurs activités manuelles sont nécessaires pour décrire le dommage exhaustivement, par exemple la prise de mesures, de photographies etc.,
- Les données acquises au cours de l'état des lieux ne peuvent être entrées directement dans le système d'information (sauf si l'utilisateur opère un dispositif mobile à connection sans fil),
- Les dispositifs mobiles classiques tels que les PDA n'autorisent que des approches basées sur les formulaires et les entrées textuelles, avec les limitations déjà décrites ci-dessus.

À l'inverse, une interaction en réalité mixte pourrait permettre à l'expert de saisir directement l'ensemble des informations pertinentes pour l'état des lieux directement depuis le logement. Il s'agirait alors de privilégier la facilité et l'intuitivité d'interaction avec le système. De plus, la continuité d'interaction entre éléments numériques et monde physique des systèmes de réalité mixte réduirait la charge de travail des utilisateurs en transférant les tâches de localisation au système. Dans le cas de l'état des lieux, le fait de localiser aisément, voire automatiquement, les dommages, augmenterait indiscutablement l'efficacité de ce processus métier, en plus de répondre aux prescriptions ergonomiques.

À l'issue de cette activité, le spécialiste IHM et l'ergonome produise une affectation des types d'interaction pour chaque PMC du processus métier étudiée. L'ergonome complète cette affectation par des prescriptions ergonomiques plus spécifiques aux types d'interaction, si nécessaire.

S'appuyant sur le type d'interaction défini pour chaque PMC, les prescriptions ergonomiques et la description des PM et PMC (notamment les scenarii structurés), l'ergonome réalise une **sélection de panels représentatifs d'utilisateurs**, en prévision des évaluations impliquant les utilisateurs. Dans le contexte de la réalisation d'un état des lieux, il s'agit des profils correspondant aux experts commissionnés par les agences immobilières : généralement des adultes de 25 à 50 ans, sans handicap moteur notable.

Enfin, l'équipe de développement priorise le développement des processus métier composants (activité d'**affectation de priorités aux PMC**), selon des critères de maximisation de la valeur délivrée aux décideurs, d'évaluation des risques et coûts associés au développement.

5.5 Spécification organisationnelle et interactionnelle des besoins

La phase de spécification des besoins organisationnels et interactionnels (c.f figure 5.6) doit déterminer le « qui fait quoi et quand » du projet. Il s'agit, à partir des descriptions conceptuelles, de détailler le comportement des acteurs internes de façon à réaliser concrètement les objectifs de chaque processus métier. Nous avons étendu cette phase pour y inclure la spécification de l'interaction, à partir du choix de style d'interaction effectué durant la phase précédente.

5.5.1 Décomposition organisationnelle des processus métier composants

L'activité de décomposition organisationnelle des processus métier composants découpe ces derniers en **activités** (voir chapitre précédent). Nous rappelons en figure 5.7 le diagramme d'activités UML résultant de cette décomposition.

Comme indiqué au chapitre précédent, le diagrammes d'activités UML produit par cette

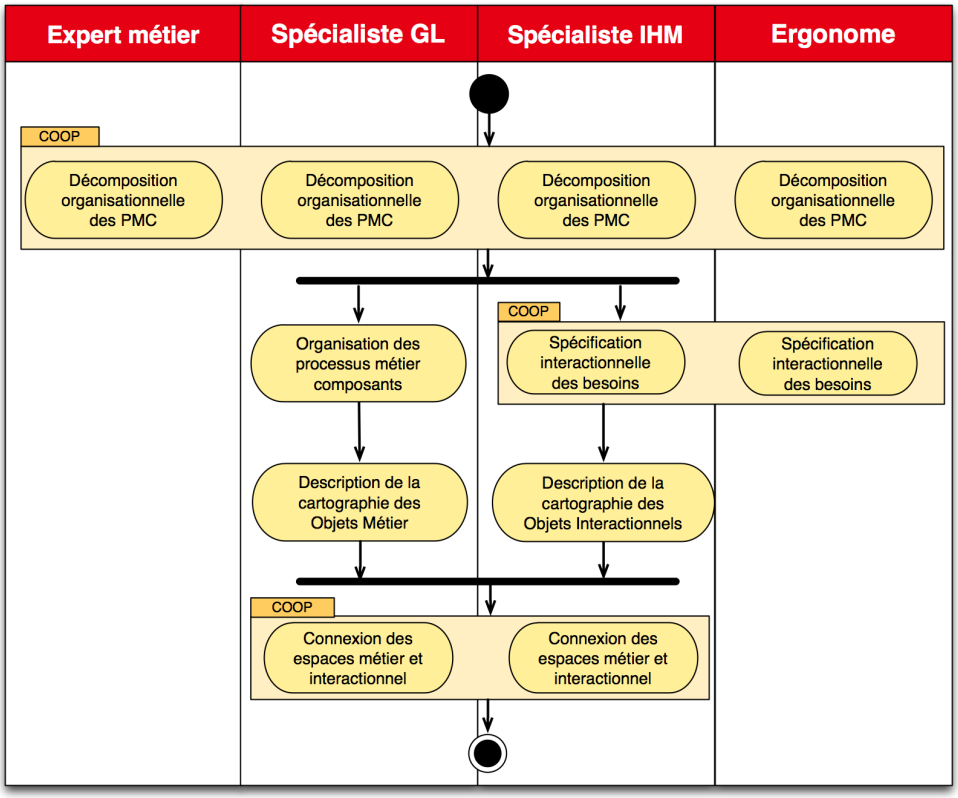


Figure 5.6 – Phase de spécification organisationnelle et interactionnelle des besoins

activité constitue une base organisationnelle pour les spécifications fonctionnelles et interactionnelles. En effet, l'une des originalités fondamentales de notre méthode apparaît dans cette phase, dans laquelle les spécialistes des domaines métier et interaction se séparent pour élaborer leur propre vision de l'application, en utilisant leurs propres pratiques, à partir d'une interprétation commune de l'organisation du métier et des utilisateurs (acteurs internes et externes) du système. Nous promouvons ainsi délibérément la construction de deux visions du système, toutes deux valables en termes des aspects sur lesquels elles sont centrées (par exemple, les fonctionalités pour le spécialiste GL, l'ergonomie pour le spécialiste IHM et l'ergonome).

Par ailleurs, ce modèle, représentant les activités principales réalisées dans le contexte de chaque processus métier, doit être validé par les utilisateurs et les décideurs.

5.5.2 Organisation des processus métier composants

Cette activité, réalisée par le spécialiste GL, extrait les activités informatisées du diagramme d'activités précédemment réalisé, pour les factoriser et les organiser sous forme de cas d'utilisation (c.f figure 5.8). Ces derniers sont alors regroupés dans un paquetage UML, correspondant au processus métier, et éventuellement complétés. Dans l'exemple ci-dessous, l'activité informatisée « Comparer élément à EdL courant » est raffinée en activités de consultation des données d'un dommage existant et de l'historique des états des lieux passés. De même, l'activité « Gérer dommage » est raffinée en activités de création de dommage et

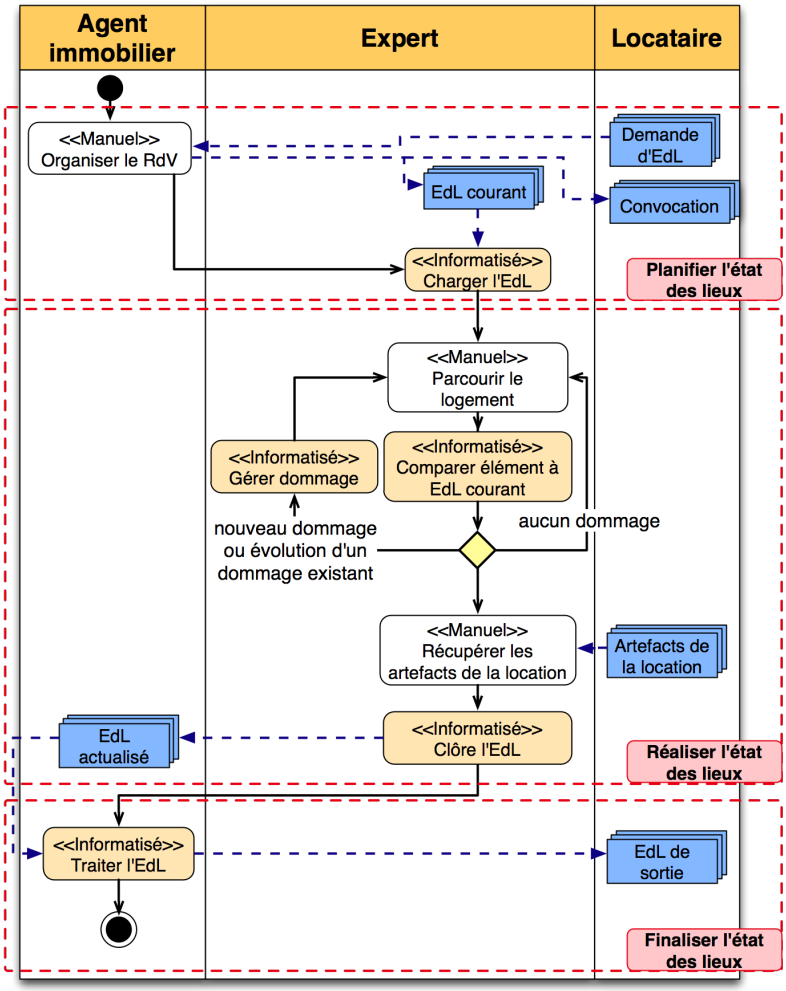


FIGURE 5.7 – Diagramme d'activités de la décomposition organisationnelle des processus métier composants, pour le processus métier « Gestion des états des lieux »

de renseignement des propriétés d'un dommage.

5.5.3 Description de la cartographie des Objets Métier

En conclusion de la spécification fonctionnelle du métier, le spécialiste GL déduit des descriptions du processus métier et des cas d'utilisation un cartographie des Objets Métier du projet. Un Objet Métier Processus, reprenant le nom du paquetage UML (ici, « Réaliser l'état des lieux » – *Realize inventory of fixtures*) précédemment décrit, est tout d'abord intégré à la cartographie (c.f figure 5.9). Les concepts centraux identifiés au cours des spécifications sont représentés sous forme d'Objets Métier Entités (dans l'exemple, ce sont les concepts d'état des lieux – *inventory of fixtures* – et de dommage – *damage* . Ces derniers sont ensuite reliés entre eux et à l'Objet Métier Processus à l'aide de relations d'utilisation (*use*). Les détails conceptuels de ce modèle sont présentés au chapitre 3.

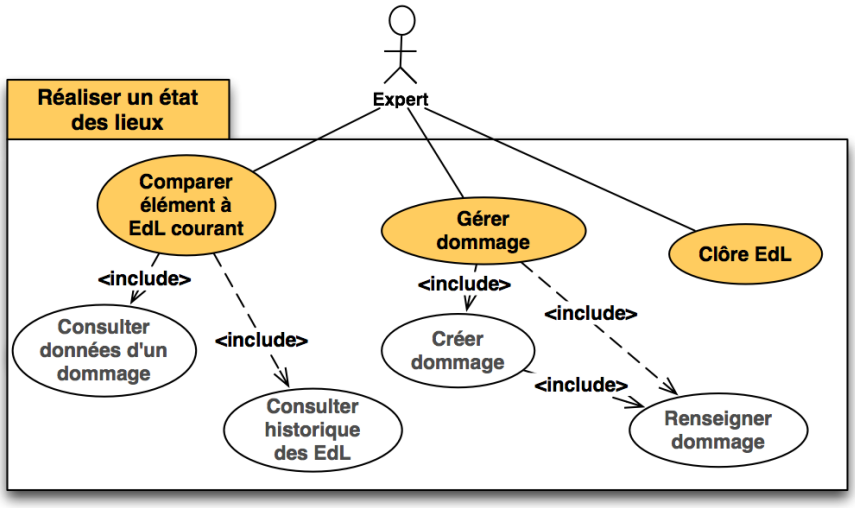


Figure 5.8 – Cas d'utilisation pour le PMC « Réalisation d'un état des lieux »

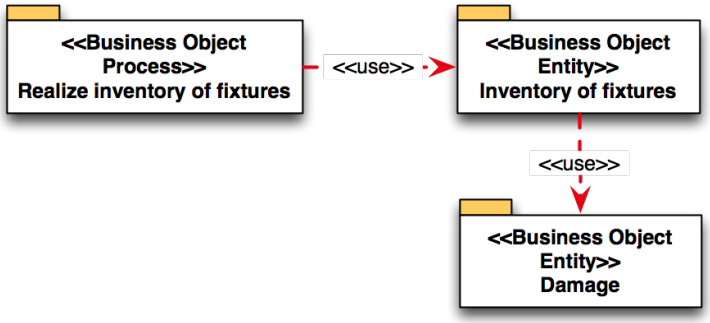


Figure 5.9 – Modèle des Objets Métier initial

5.5.4 Spécification interactionnelle des besoins

L'activité de spécification interactionnelle des besoins reprend les prescriptions ergonomiques, l'organisation des processus métier composants manuels et informatisés et le choix du type d'interaction, pour décrire l'interface homme-machine du futur système. Cette activité se focalise sur les acteurs internes du système, impliqués dans la réalisation concrète des processus métier et considérés à ce titre comme les utilisateurs de l'application en cours de développement. C'est une activité dense, dont nous présentons les sous-activités en figure 5.10. Les produits de cette activité pertinents pour les évaluations utilisateur sont regroupés dans le « Dossier de l'interaction ».

5.5.4.1 Description des scenarii abstraits

Le diagramme d'activités UML et les descriptions des processus métier composants élaborés précédemment peuvent s'avérer être insuffisants pour envisager correctement les attentes des utilisateurs, ainsi que le contexte dans lequel ils interagissent avec le système. Par conséquent, nous proposons de raffiner le diagramme d'activités UML des processus métier composants manuels et informatisés, pour décrire des scenarii abstraits projetés. Le

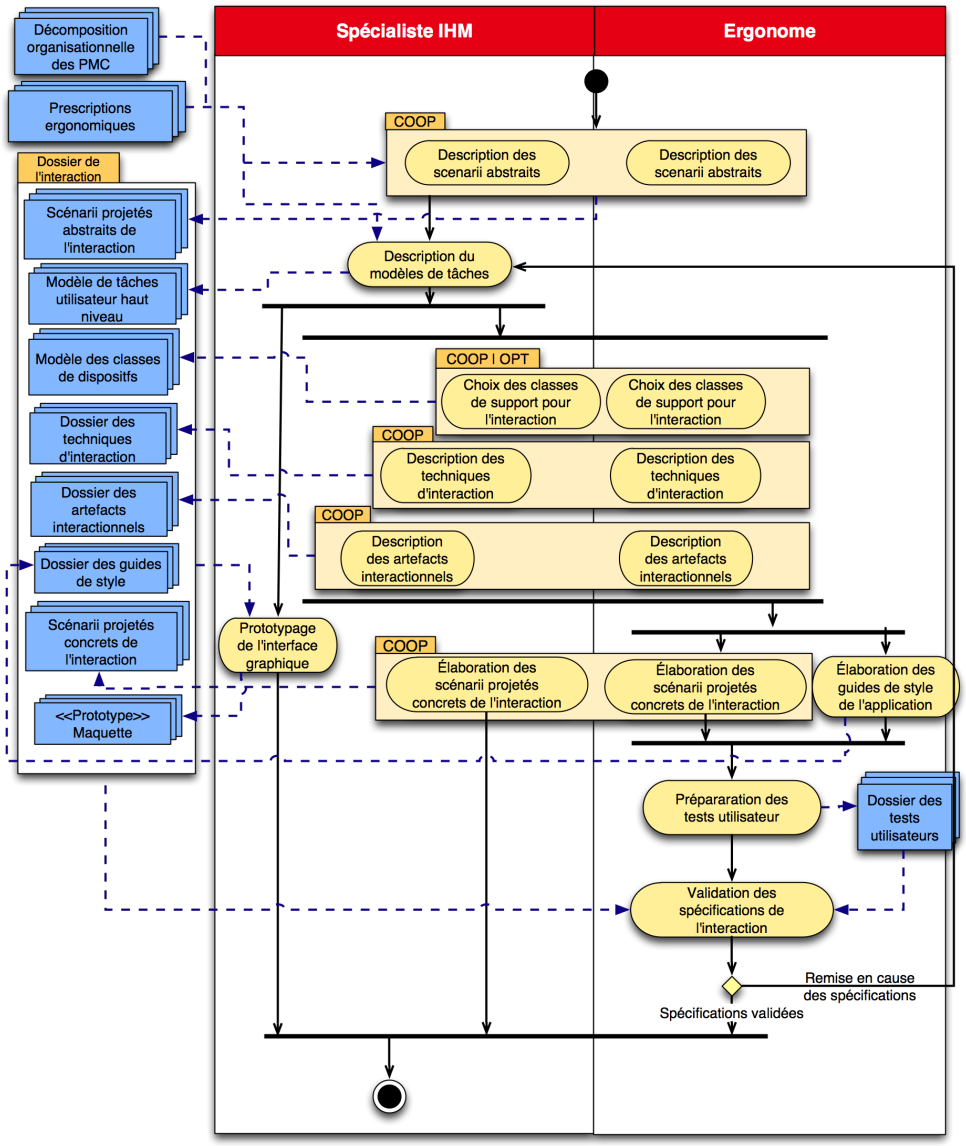


FIGURE 5.10 – Activités de spécification interactionnelle des besoins

périmètre de ces derniers ne couvre qu'un petit nombre de processus métier composants à la fois : il s'agit en effet de capturer quelques situations d'interaction, qui seront reprises pour l'élaboration des tests utilisateur. Cette activité implique une coopération entre le spécialiste IHM et l'ergonome, lequel s'assure que les prescriptions ergonomiques décrites durant la phase de spécification conceptuelle des besoins sont respectées.

L'exemple de scénario projeté abstrait (c.f tableau 5.5) illustre une activité utilisateur comprise dans le processus métier composant « Gérer dommage ».

5.5.4.2 Description du modèle de tâches

Il n'est pas rare que les interactions entre l'utilisateur et le système présentent une forte complexité, intègrant tâches parallèles, fréquentes, en recouvrement partiel ou total, tâches

Tableau 5.5 – Extrait du scénario projeté abstrait de la tâche « Créer un dommage »

(…) L'expert rentre dans une *pièce*. L'*ancien état des lieux* ne lui indique aucun dégât ou usure à vérifier pour cette pièce. Il parcourt la pièce et remarque néanmoins un tâche sombre sur l'un des murs. Il décrit le dommage observé, et prend éventuellement un *instantané* (photo ou vidéo) du dommage dans son contexte (…).

critiques ou optionnelles. Ces quelques exemples dépassent l'expressivité des diagrammes d'activités UML. Subséquemment, nous proposons la description d'un modèle de tâches à haut niveau d'abstraction, lequel reprend les processus métier composants du diagramme d'activités, tout en complétant ces derniers avec la sémantique d'enchaînement et les qualificateurs propres aux formalismes tels que CTT ou K-MAD (c.f chapitre 1). Cette approche complémente la description des activités informatisées et se rapproche des préoccupations de l'interaction homme-machine. La figure 5.11 décrit ainsi, à titre d'exemple, les tâches à haut niveau d'abstraction correspondant au processus métier composant « Réalisation d'un état des lieux ».

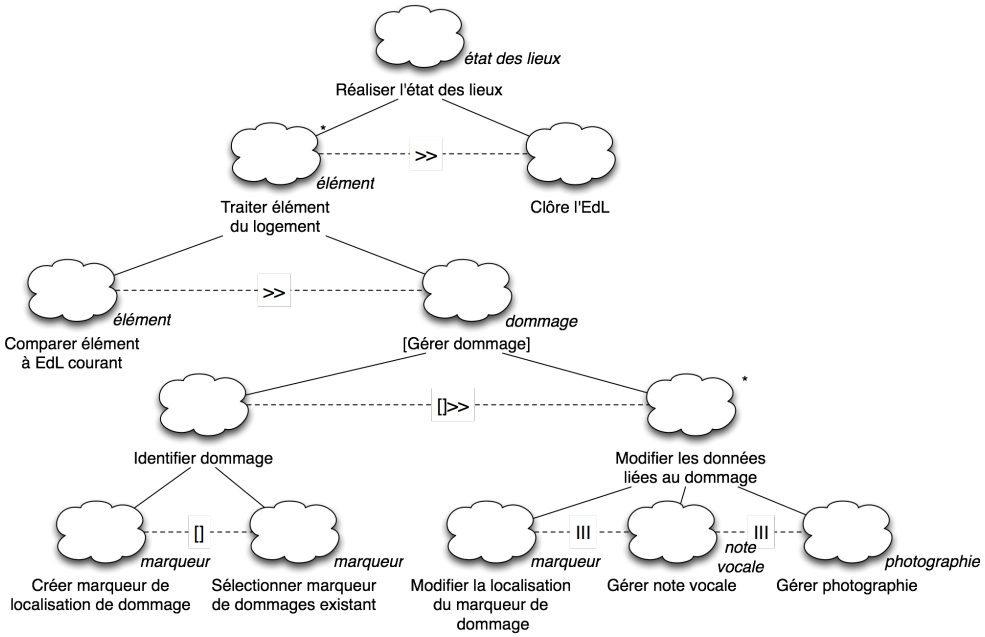


Figure 5.11 – Tâches utilisateur de haut niveau pour la réalisation d'un état des lieux

5.5.4.3 Élaboration des artefacts, techniques et dispositifs d'interaction

À partir des scenarii et/ou des modèles de tâches, le spécialiste IHM et l'ergonome se focalisent sur trois activités interdépendantes de la future interaction : la **description des artefacts d'interaction**, la **description des techniques d'interaction**, ainsi que le **choix des classes de support pour l'interaction**. Cette dernière activité est optionnelle, n'étant nécessaire que pour la construction d'interfaces de réalité mixte. Par souci de concision, nous limitons l'illustration de ces activités aux tâches de gestion d'un dommage.

Par ailleurs, précisons que les activités décrites et illustrées ci-dessous peuvent être complé-

tées par des approches de conception plus systématiques, telles que la sélection de patrons d'utilisabilité de la méthode UPi (c.f chapitre 2), l'utilisation des recommandations ergonomiques décrite par Charfi et al. [24] (c.f chapitre 1), ou encore le système de patrons pour la conception d'interfaces multimodales que nous avons présenté dans Godet-Bar et al. [54].

Afin de minimiser la quantité d'informations présentée à l'utilisateur, nous découpons l'interaction en deux parties : à distance d'un dommage et proche d'un dommage. Lorsque l'utilisateur est distant d'un dommage, seul un marqueur indique sa présence, sous forme d'une cible. Lorsqu'il se rapproche d'un dommage, un plus grand nombre d'informations apparaît en transparence, telles que des photographies des états antérieurs, des indicateurs représentant l'existence de notes vocales attachées au dommage. De plus, dans cette situation, l'utilisateur peut rajouter des informations concernant le dommage (nouvelles notes vocales ou photographies) ou supprimer le dommage. À tout instant, l'utilisateur peut rajouter un dommage.

La description des tâches utilisateur à haut niveau d'abstraction nous permet de définir un premier ensemble de concepts centraux autour desquels nous construirons l'interaction : les **marqueurs**, permettant de localiser les dommages, les **notes vocales** (plus particulièrement, des objets interactifs permettant l'ajout, la suppression et la consultation d'éléments audio) et les **photographies** (de même, des objets interactifs permettant l'ajout, la suppression et la consultation de photographies). Si les notes vocales et les photographies procèdent directement du scénario abstrait, les marqueurs sont déduits des propriétés physiques des dommages perçues et signalées par l'utilisateur (notamment, la localisation de ces derniers).

D'autres concepts peuvent émerger au cours de la conception de l'interaction, sans lien avec ces scenarii. L'élaboration d'une interaction en réalité mixte requiert ainsi d'identifier la position de l'utilisateur dans le monde physique. Dans le contexte de l'état des lieux augmenté, il est nécessaire de repérer la position de l'utilisateur par rapport au logement. D'autre part, dans la mesure où les marqueurs de dommages peuvent être positionnés en tout lieu du logement, et plus particulièrement sur les murs, le plafond etc., la représentation numérique du logement doit être tridimensionnelle. Subséquemment, deux nouveaux concepts interactionnels émergent : le **plan 3D** du logement et l'**avatar**, représentant la position et l'orientation de l'utilisateur dans le logement.

En termes de classes de dispositifs, nous envisageons d'utiliser pour l'essentiel : des **lunettes de réalité augmentée** (lunettes semi-transparentes intégrant des surfaces d'affichage) intégrant des gyroscopes (permettant de suivre le regard de l'utilisateur) et un **dispositif mobile tactile** disposant d'accéléromètres (permettant de suivre la position du dispositif ainsi que des interactions gestuelles) et d'un capteur vidéo (permettant entre autre la prise de photographies).

Nous résumons l'organisation des dispositifs et des flux d'informations principaux de l'application dans un modèle ASUR du système (c.f figure 5.12). Nous rappelons que les diagrammes ASUR permettent de représenter les éléments numériques et physiques du système impliqués dans l'interaction homme-machine, ainsi que les dispositifs utilisés et les éventuelles relations mécaniques qu'ils entretiennent.

La figure 5.12 ne détaille qu'une partie des flux d'informations, et se focalise sur le concept interactionnel de marqueur, ainsi que sur les dispositifs nécessaires à sa représentation et sa manipulation. L'utilisateur, identifié comme l'expert, porte un dispositif de vision augmentée (la relation « == » dénote le lien physique ou mécanique entre dispositifs et/ou utilisateur). Ce

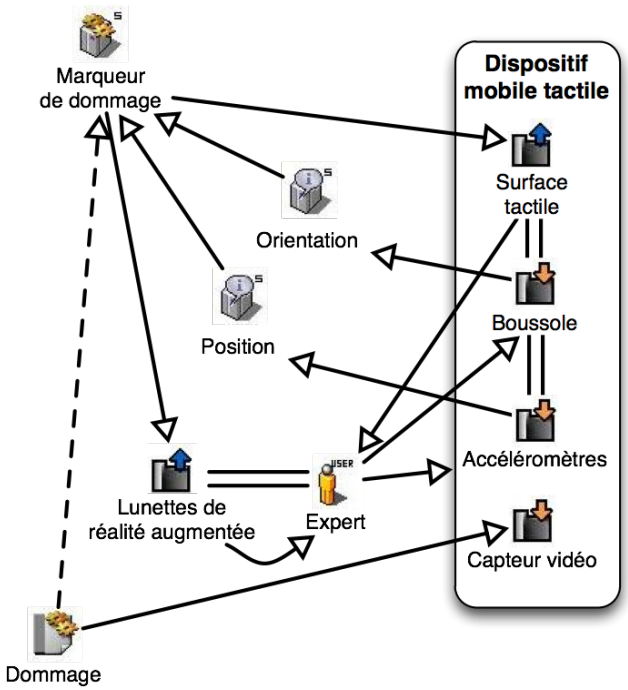


Figure 5.12 – Diagramme ASUR du système

dispositif affiche les marqueurs, ainsi que des informations concernant les dommages, sous forme de notes vocales virtuelles. La relation « –> » symbolise le transfert d'informations (physique ou numérique) entre les différents éléments du modèle. Le dispositif d'affichage mobile, lié à l'entrée tactile, fournit des informations concernant la position de l'utilisateur dans le logement (grâce au plan 3D), grâce à un système de positionnement suivant l'expert au fil de ses déplacements, ainsi que l'interface de contrôle du système (menus etc.). Les marqueurs sont des représentations numériques (relation « – – > ») de dommages physiques.

Quant aux techniques d'interaction, celles-ci sont également formalisées à l'aide de modèles de tâches utilisateur, cette fois-ci à plus bas niveau d'abstraction. Comme précédemment, nous avons recours au formalisme ConcurrTaskTree [93], dont nous utilisons les représentations des tâches physiques, de communication et système. La figure 5.13 décrit la technique d'interaction adoptée pour la tâche « Modifier la localisation du marqueur de dommage » (les opérateurs de séquencement ont été omis au niveau le plus concret, par souci de lisibilité).

Les techniques d'interaction que nous envisageons d'utiliser s'appuient sur le concept de « loupe » numérique mentionné plus haut. L'utilisateur utilise ainsi son dispositif mobile – celui-ci affichant comme par transparence le monde physique grâce à son capteur vidéo – pour « viser » les zones du logement sur lesquelles il veut ajouter ou supprimer un marqueur de dommage, ou bien ajouter une photographie ou une note vocale. Un exemple d'une telle interaction est présentée en figure 5.14.

Il est également possible à l'utilisateur de redimensionner ou de repositionner le marqueur grâce à des commandes tactiles (gestes de pincement ou de déplacement sur la surface tactile). Une fois la position du marqueur satisfaisante, celui-ci est verrouillé en appuyant sur un bouton du dispositif mobile. Un nouvel appui sur ce bouton déverrouille le marqueur et permet de le dimensionner et positionner à nouveau.

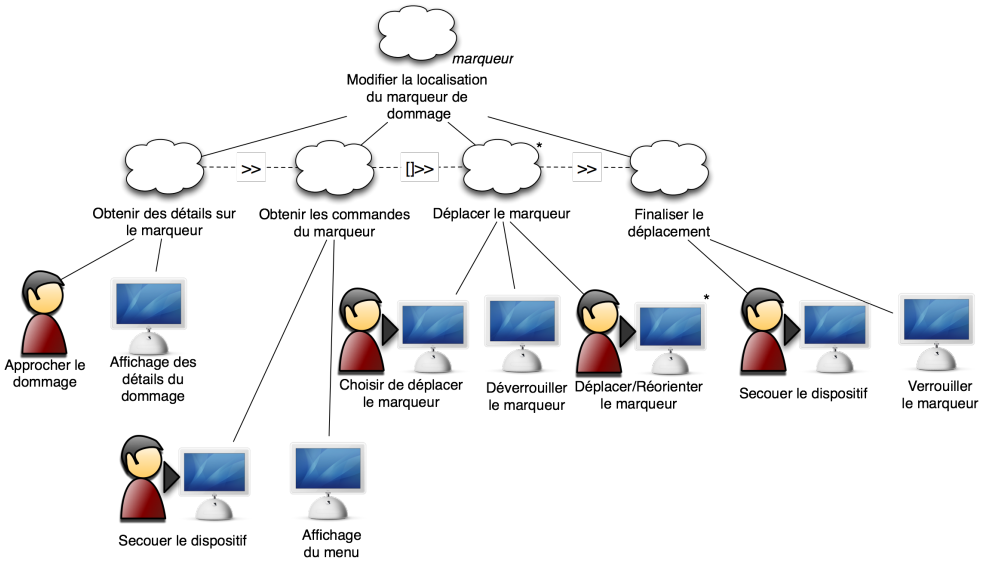


FIGURE 5.13 – Technique d'interaction pour la tâche « Modifier la localisation du marqueur de dommage »

FIGURE 5.14 – Exemple d'interaction au travers d'une « loupe » numérique

Lorsque l'utilisateur désire consulter les données des dommages, il lui suffit de s'approcher de l'un d'eux : le système affiche alors, en plus du marqueur, les notes vocales et photographies en transparence sur les lunettes de réalité mixte. Les notes vocales ou photographies directement regardées par l'utilisateur (c'est-à-dire, dans l'axe des lunettes de réalité mixte) sont mises en surbrillance. Le fait d'appuyer deux fois de suite sur la surface tactile du dispositif mobile déclenche la lecture de la note, via son haut-parleur intégré, ou agrandit la photographie, sur les lunettes de réalité augmentée. De plus, la liste des notes et photographies est affichée en redondance des données affichées sur les lunettes de réalité augmentée, afin de permettre une consultation directe de celles-ci via des commandes tactiles.

Tableau 5.6 – Extrait du scénario projeté concret de la tâche « Créer un dommage »

(…) L'expert entre dans une *pièce*. Le *précédent état du logement* est automatiquement chargé alors qu'il entre dans la pièce. Aucun *marqueur de dommages* antérieur n'est affiché sur les *lunettes de vision augmentée*. Il se déplace dans la pièce et remarque une tache sombre sur l'un des murs. Il crée un nouveau *marqueur de dommages* et le positionne, l'oriente et le dimensionne à l'aide du *dispositif mobile tactile*, afin de le localiser précisément sur la tache. Puis, il verrouille le marqueur et y attache une *note vocale*, à l'aide du *microphone*.

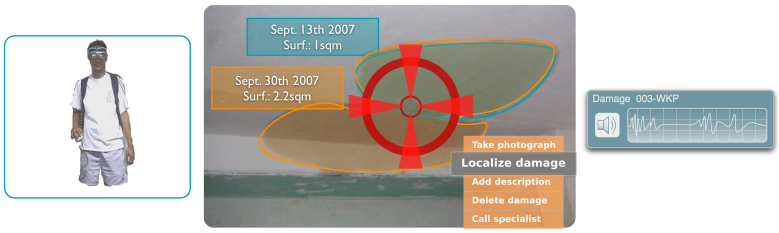


Figure 5.15 – Prototype de l'application

5.5.4.4 Description des scenarii projetés concrets

Les scenarii projetés concrets reprennent les scenarii projetés abstraits élaborés précédemment, et y intègrent les artefacts interactionnels, les techniques d'interaction et les classes de support pour l'interaction. À titre d'exemple, le tableau 5.6 raffine le scénario décrit au tableau 5.5. Ces scenarii garantissent d'une part que toutes les situations d'interaction couvertes par les scenarii abstraits sont couvertes ; d'autre part ils établissent le périmètre des tests utilisateur à mener pour valider l'IHM.

5.5.4.5 Élaboration des guides de style de l'application

Au fil de la description de la spécification interactionnelle des besoins, l'ergonome organise des recommandations et règles pour l'IHM en cours de construction, à partir des produits des différentes activités. Il pourra s'agir de chartes graphiques, de contraintes liées aux spécificités culturelles (registre de langage, symboles, couleurs) et/ou physique (usage adéquat des modalités) des utilisateurs. Les guides de style ainsi produits servent alors de référence pour la cohérence modale, culturelle et conceptuelle de l'IHM.

5.5.4.6 Prototypage itératif

Tout au long des activités d'élaboration de l'interaction du futur système, nous recommandons la construction de prototypes logiciels (sous forme de présentations interactives, de maquettes…) mettant en jeu les produits de la spécification de l'interaction. Au-delà de la possibilité d'explorer différentes solutions de conception, les prototypes permettent également aux ergonomes de préparer les évaluations qui valideront la spécification interactionnelle.

La figure 5.15 présente un prototype de l'interface telle qu'affichée sur le dispositif de vision augmentée de l'utilisateur.

5.5.4.7 Évaluation ergonomique

En se basant sur les scenarii projetés concrets, les prototypes, et les panels d'utilisateurs représentatifs (c.f section 5.4), l'ergonome élabore des tests de validation impliquant tous les produits de la spécification interactionnelle. Des itérations de développement de la spécification interactionnelle sont réalisées jusqu'à ce que celle-ci soit validée pour l'ensemble des produits qui lui sont attachés.

5.5.5 Description de la cartographie des Objets Interactionnels

Cette dernière activité de la spécification interactionnelle des besoins propose d'organiser les concepts de l'interaction en un diagramme d'Objets Interactionnels, tel que présenté au chapitre précédent. Ces derniers sont en effet suffisamment denses en termes sémantiques, et de granularité suffisamment large, pour constituer des Objets Interactionnels Entités adaptés. On recense ainsi cinq Objets Interactionnels Entités : « Marqueur » (*Marker*), « Plan 3D » (*3D Plan*), « Avatar », « Zone d'affichage » (*Display zone*) et « Note vocale » (*Vocal note*). La spécification interactionnelle des besoins permet au spécialiste IHM d'identifier un Objet Interactionnel Processus (ici, il s'agit de gérer une scène de réalité augmentée, en trois dimensions), autour duquel sont agencés les Objets Interactionnels Entités. Nous en présentons la cartographie en figure 5.16.

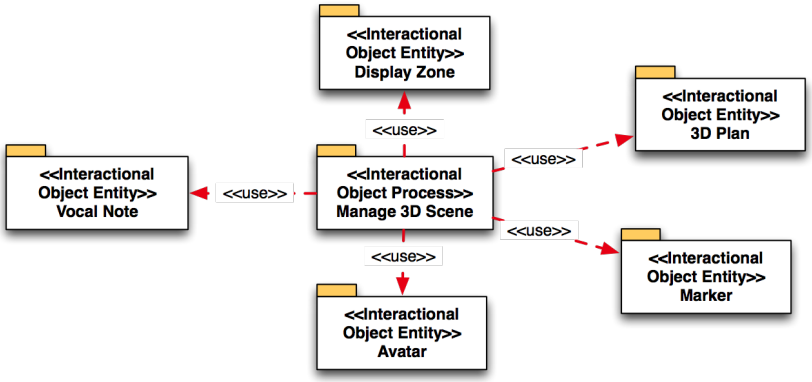


FIGURE 5.16 – Modèle des Objets Interactionnels

La dernière activité de cette phase concerne la connexion entre les Objets Métier et les Objets Interactionnels. Nous avons pour cela introduit dans la méthode Symphony une nouvelle relation de dépendance nommée « Représente », qui permet d'expliciter des appariements implicites jusque-là (c.f section 4.3) : certains Objets Interactionnels apparaissent comme des projections dans l'espace interaction de concepts métier. Toutefois, la description de ces relations peut induire des évolutions de l'espace métier. C'est ce que nous décrivons dans la section suivante.

5.6 Une activité de coopération pour envisager l'évolution du métier

Nous étudions dans cette section comment des choix d'interaction adoptés par l'ergonome et le spécialiste IHM peuvent déclencher une évolution de l'espace métier. Cette étude

commence au niveau de raffinement des spécifications le plus bas (niveau organisationnel), pour remonter jusqu'au niveau de la définition du métier (niveau intentionnel). La figure 5.17 résume les différentes activités d'évolution (partie gauche de la figure), en vis-à-vis des phases du développement impactées par leur réalisation. Les produits générés lors de la mise en œuvre de ces activités ont pour vocation d'être réintégrés dans le flot d'artefacts du cycle de développement.

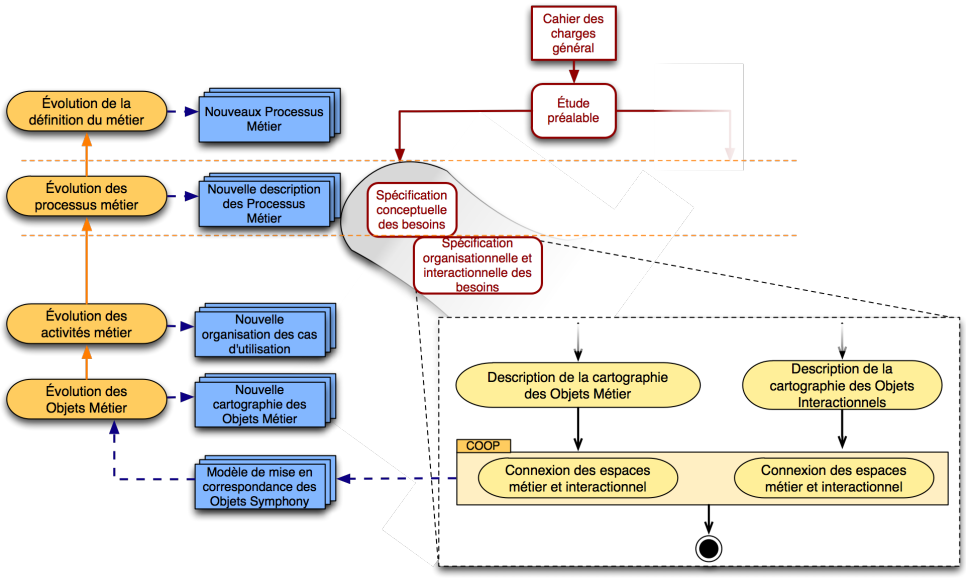


Figure 5.17 – Sous-processus d'évolution du métier

5.6.1 Une activité de mise en relation de modèles

L'objectif initial de cette activité est de décrire les relations « Représente » entre Objets Interactionnels et Objets Métier. Cette relation un-vers-plusieurs décrit explicitement de quelle façon certains concepts métier sont projetés dans l'espace interaction. Par exemple, la notion de dommage apparaît dans l'espace interaction au travers des notions de « Marqueur » et de « Note vocale ».

Une comparaison entre le modèle d'Objets Métier (voir figure 5.9) et le modèle d'Objets Interactionnels (voir figure 5.16) révèle différents niveaux de granularité entre concepts métier et concepts interactionnels. Par exemple, le « Marqueur », qui intègre des attributs de position, taille, un comportement particulier, un cycle de vie et une représentation visuelle est plus dense que ce qu'il prétend représenter, c'est-à-dire le « Dommage », qui ne contient que des attributs liés à sa position (par exemple une pièce du logement) et une description textuelle. Par conséquent, les acteurs du développement peuvent envisager de résorber ce déséquilibre de densité de services proposés.

Les paragraphes suivants offrent des propositions d'évolution de l'espace métier, induites par les choix d'interaction, aux niveaux organisationnel (Objets Métier et activités), conceptuel (processus métier) et intentionnel (définition du métier).

5.6.2 Évolution des Objets Métier

Il s'agit à ce niveau de vérifier si le modèle des Objets Interactionnels prend en charge des préoccupations qui pourraient correspondre à des responsabilités du métier. L'analyse vise à décider si le système d'information de l'organisation tirerait un bénéfice d'un transfert et d'une adaptation de ces données dans l'espace métier.

Par exemple, les systèmes de réalité mixte intègrent souvent des données de localisation (le « Plan 3D » dans notre état des lieux augmenté), afin de localiser l'utilisateur ou bien des artefacts physiques dans un environnement 3D superposé au monde physique. Un transfert de compétences envisageable consisterait à étendre les responsabilités des Objets Métier pour y inclure la gestion des plans d'architecte des logements. L'Objet Interactionnel « Plan 3D » demeurerait responsable de la gestion de la représentation visuelle du plan d'architecte. Ainsi dans notre cas d'étude, nous pourrions intégrer le plan du logement dans l'espace métier (et donc dans le système d'information) à l'aide d'un nouvel Objet Métier Entité « Logement » (*Housing*). La représentation du logement dans l'application de réalité mixte résulterait d'une transformation du plan en coordonnées 3D.

Le concept de logement n'est pas le seul susceptible de subir une transformation causée par les choix d'interaction. En effet, l'Objet Métier « Dommage » est également un concept peu dense dans le PM « Réalisation d'un état des lieux » et plus large dans le modèle des Objets Interactionnels (étant représenté par les Objets Interactionnels « Marqueur », « Note vocale » et « Zone d'affichage »).

Une fois de plus, l'intégration dans l'espace métier des données gérées par ces artefacts (c'est-à-dire, les notes vocales, les photographies…) pourrait s'avérer avantageux pour d'autres processus métier. Toutefois, étant donné la granularité de ces informations, il n'est pas nécessaire de diviser l'Objet Métier Entité « Dommage » en éléments plus fins, qui les intégrera dans sa structure actuelle. La figure 5.18 restranscrit cette évolution du modèle des Objets Métier, avant et après l'activité de coopération entre les spécialistes GL et IHM.

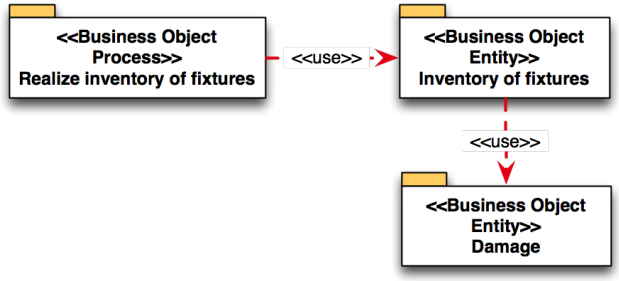
Suivant cette évolution, nous présentons en figure 5.19 la mise en relation finale entre Objets Métier et Objets Interactionnels, à l'aide de la relation « représente ». Nous constaterons à la section 5.7 que cette relation, d'abord simpliste, est ensuite raffinée en un modèle complet de communication entre les espaces métier et interaction.

5.6.3 Évolution des activités métier

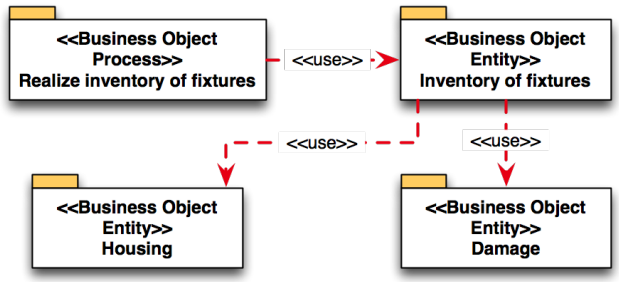
Nous avons pu voir au paragraphe précédent que les transferts de compétences entre Objets Interactionnels et Objets Métier peuvent introduire de nouveaux concepts dans l'espace métier. Par conséquent, de nouvelles activités pour collecter, organiser et utiliser ces données doivent être décrites au niveau des activités métier.

Du point de vue du processus de développement, de nouveaux cas d'utilisation pour mener ces activités font leur apparition. Subséquemment, ceux-ci affectent l'organisation des cas d'utilisation décrite au cours de la phase de spécification conceptuelle des besoins (voir section 5.4). On retrouve dans la figure 5.20 un extrait de l'organisation des cas d'utilisation avant et après l'évolution de l'espace métier.

À présent que nous avons envisagé d'intégrer photographies, notes vocales et notes tex-



(a) Avant coopération



(b) Après coopération

FIGURE 5.18 – Modèle partiel des Objets Métier

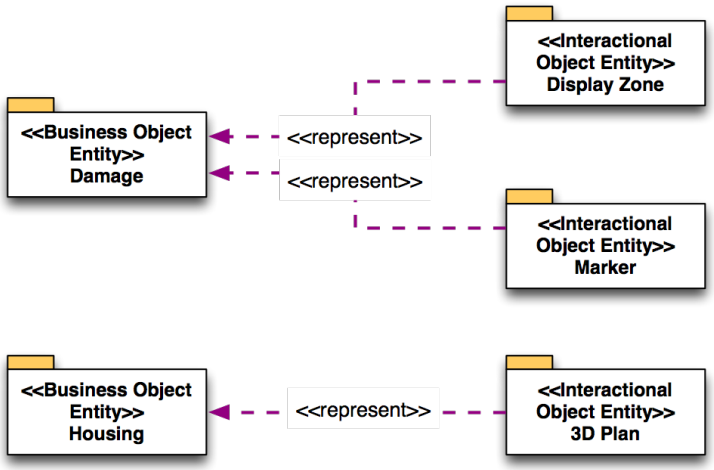
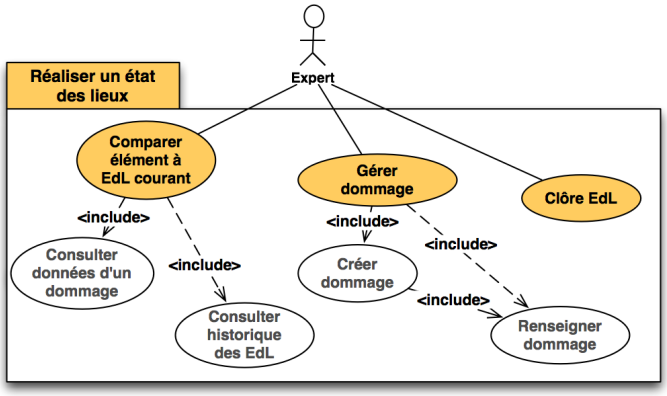
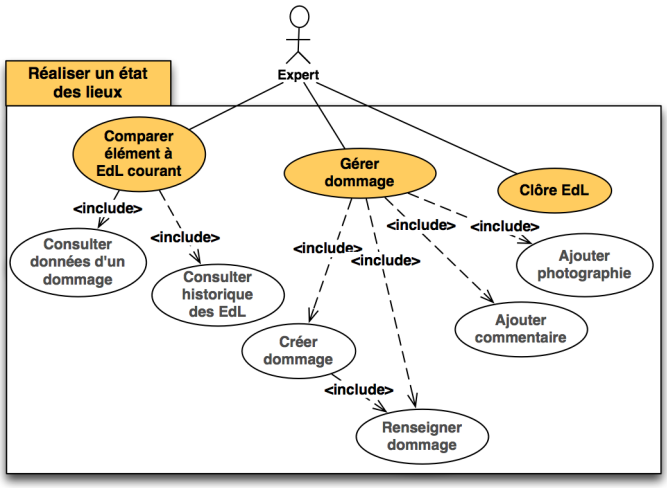


FIGURE 5.19 – Extrait du modèle de mise en relation des Objets Métier et Objets Interactionnels

tuelles au concept de dommage, dans l'espace métier, nous devons ajouter plusieurs cas d'utilisation pour gérer ces nouveaux éléments (c'est-à-dire, pour les rattacher au dommage, voir figure 5.20(b)).



(a) Avant évolution



(b) Après évolution

FIGURE 5.20 – Organisation des cas d'utilisation

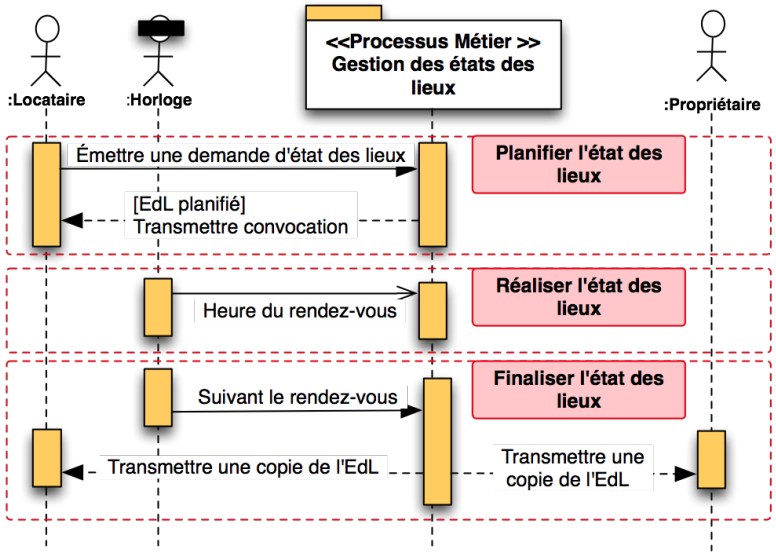
5.6.4 Évolution des processus métier

Amener l'évolution du métier à ce niveau implique de capitaliser sur les nouveaux cas d'utilisation introduits au niveau activité métier. En particulier, la mise en œuvre de ces nouvelles données par de nouveaux PMC et l'intervention de nouveaux acteurs au sein du système d'information peuvent être envisagées. Cette démarche peut permettre l'automatisation de tâches autrefois manuelles, ou bien la simplification de certains processus.

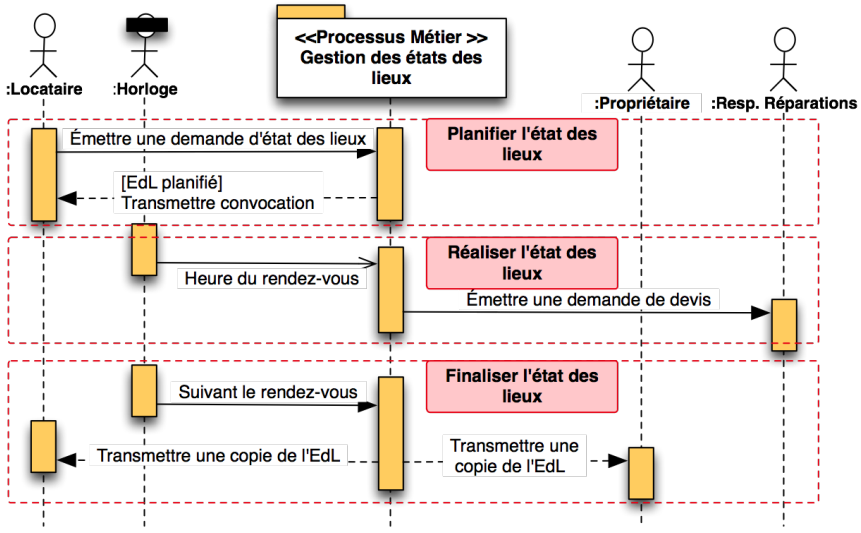
Lorsque nous considérons les cas d'utilisation ajoutés au niveau activités métier, pour l'état des lieux augmenté, nous constatons que le choix d'une interaction en réalité mixte nous permet de disposer, entre autres choses, d'un plus grand niveau de détail pour décrire les dommages.

D'autre part, parmi les activités clef de l'agence immobilière, on distinguera la gestion des réparations des logements sous sa responsabilité, ainsi que l'envoi de demandes de devis. À présent, l'expert commissionné par l'agence a accès aux précédents état des dommages, ainsi que la capacité d'intégrer immédiatement des photographies de son état actuel dans le SI. Par conséquent, il peut demander directement un devis, soit en envoyant les données idoines

au personnel de l'agence immobilière assumant le rôle de gestionnaire des réparations – et qui sont donc maintenant impliqués dans le processus métier –, ou en communiquant directement les données à l'organisation responsable des réparations (artisan, entreprise de nettoyage etc.), laquelle organisation est alors un acteur secondaire et externe. La figure 5.21 montre cette évolution du niveau processus métier du système.



(a) Avant évolution



(b) Après évolution

FIGURE 5.21 – Diagrammes de séquence du PM « Réalisation d'un état des lieux »

5.6.5 Évolution de la définition du métier

Au-delà de la réorganisation des PM, l'évolution du métier peut être poursuivie au point où de nouveaux services sont intégrés au système d'information, modifiant ainsi la définition

du métier. De nouveaux PM peuvent être ajoutés, qui requerront leur propre itération du processus de développement.

Par conséquent, de nouveaux choix d'interaction seront effectués pour ces nouveaux processus, et de nouveaux acteurs pourront être intégrés à l'espace métier.

Dans le contexte de l'état de lieux augmenté, les développeurs et décideurs peuvent s'accorder sur la mise en place de nouveaux processus, grâce à l'intégration du plan d'architecte des logements. Par exemple, l'agence immobilière peut proposer des visites virtuelles des logements à ses futurs clients.

La section suivante détaille la dernière phase du développement fonctionnel : l'analyse, qui a lieu une fois le sous-processus d'évolution stabilisé.

5.7 Analyse

Cette phase décrit le « comment » du comportement du système, plus particulièrement de quelle façon les Objets Métier et Objets Interactionnels communiquent au travers de leurs relations « utilise » et « représente » et de quelle façon ils sont structurés en termes d'attributs et de services. Le premier aspect de cette étude est détaillé dans une analyse dynamique du système, tandis que le second aspect est élaboré au cours d'une analyse statique. Puis, la sémantique de la communication entre les espaces métier et interactionnel, matérialisée par la relation « représente », est définie. On notera que les aspects techniques et architecturaux du système ne sont pas pris en compte à ce niveau du développement.

5.7.1 Analyse dynamique

Du point de vue du spécialiste GL, l'analyse dynamique consiste à raffiner les cas d'utilisation précédemment identifiés en scenarii détaillés et diagrammes de séquences UML. Ces descriptions ont pour objectif de compléter la construction des Objets Métier, en précisant les services (c'est-à-dire, l'interface) qu'ils doivent proposer.

La figure 5.22 illustre quelques l'enchaînement d'appels pour la création d'une instance de description de dommage, dans les classes intégrées dans les paquetages « Réaliser état des lieux » (*RealizeInvOfFixtures*), « État des lieux » (*Inventory of Fixtures*) et « Dommage » (*Damage*). Dans cet exemple, l'utilisation de la classe rôle (c'est-à-dire, « DommageLocalisé ») permet d'améliorer la réutilisabilité de l'Objet Métier « Dommage ». En effet, la sémantique de la localisation des dommages est propre à l'application, dans mesure où elle utilise le concept de pièce pour placer le dommage sur le plan d'architecte. Le fait de transférer tous les appels impliquant la position du dommage dans l'Objet Métier Processus (qui est, par définition, spécifique de l'application) permet ainsi de réduire la dépendance de l'Objet Métier Entité « Dommage » vis-à-vis du système.

Du point de vue du spécialiste IHM, l'analyse dynamique correspond à la formalisation du cycle de vie des Objets Interactionnels. Des diagrammes d'états UML sont utilisés pour décrire les objets ayant un cycle de vie suffisamment complexe. De plus, les scenarii concrets décrits durant la spécification de l'interaction sont raffinés en scenarii détaillés et diagrammes de séquences UML, similairement à l'analyse dynamique des Objets Métier. La figure 5.23 décrit ainsi la création d'un Objet Interactionnel Entité « Marqueur », au travers

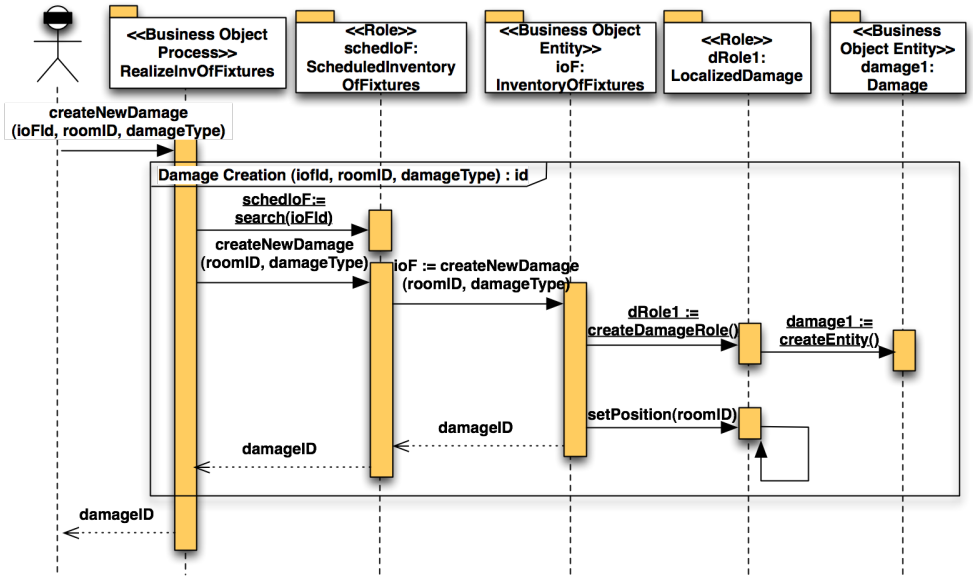


FIGURE 5.22 – Extrait de l'analyse dynamique de l'espace métier

d'une classe « Marqueur Localisé » intégrée dans l'Objet Interactionnel Processus « Gestion scène 3D ».

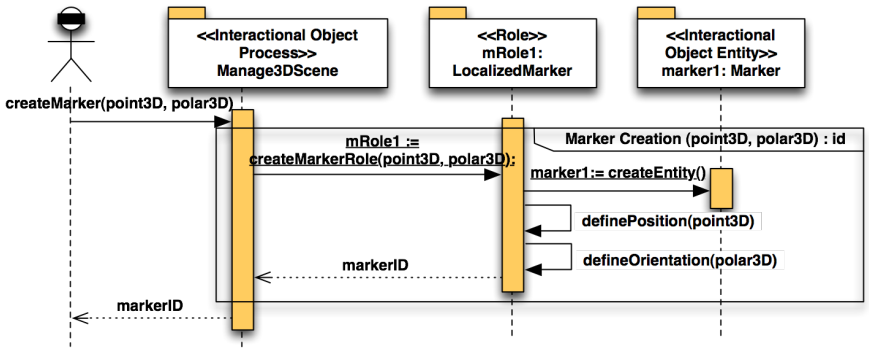


FIGURE 5.23 – Extrait de l'analyse dynamique de l'espace interactionnel

5.7.2 Analyse statique

L'analyse statique permet de recenser les services employés lors de l'activité d'analyse dynamique, et de les organiser et structurer sous forme d'Objets Symphony de niveau analyse. Ces derniers correspondent à des paquetages tripartites, ainsi que décrit à la section 4.3.

La figure 5.24 expose quelques uns des Objets Métier et Objets Interactionnels décrits dans ce chapitre. À la différence de l'exemple présenté au chapitre précédent, on notera que l'Objet Métier « État des lieux » (*Inventory of fixtures*) emploie un rôle « HousingRole », liée à l'Objet Métier « Logement » (*Housing*), reflétant ainsi l'évolution de l'espace métier décrite précédemment. Conformément à l'exemple du chapitre précédent, la relation « Représente » lie le rôle « LocalizedMarker » avec l'Objet Métier « Inventory of Fixtures ».

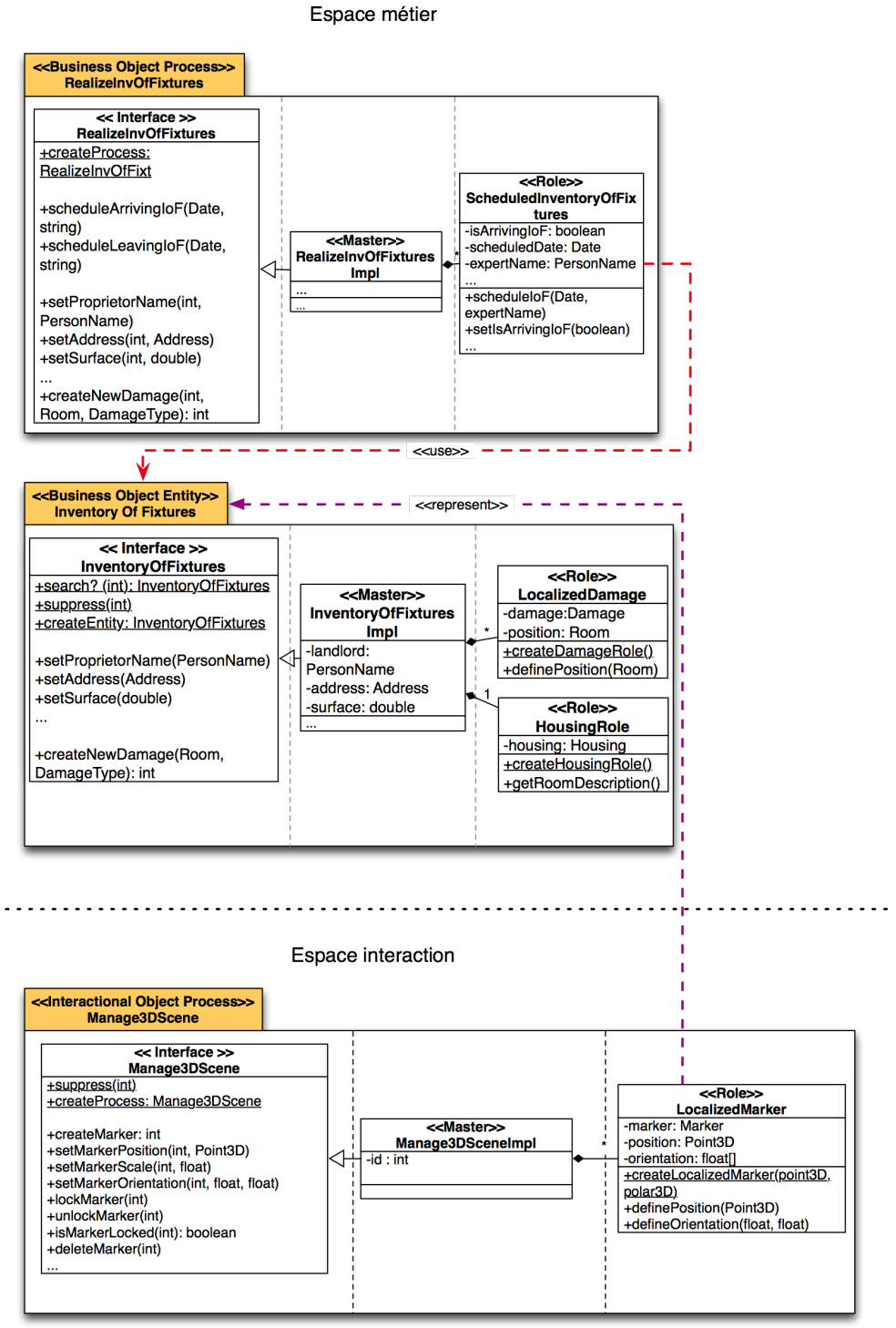


FIGURE 5.24 – Extrait du modèle d'analyse des Objets Symphony (espaces métier et interactionnel)

5.7.3 Description des relations entre les espaces métier et interactionnel

À ce niveau du développement, les espaces métier et interactionnel sont décrits exhaustivement, indépendamment des technologies d'implémentation. Le formalisme utilisé pour décrire les Objets Symphony est cohérent avec les préoccupations des analystes programmeurs, qui interviennent à partir de l'analyse pour envisager l'implémentation du système. À présent, les spécialistes GL et IHM doivent détailler la sémantique des relations « représente » identifiées durant les spécifications.

Nous rappelons que cette activité consiste à annoter les méthodes des Objets Symphony, à l'aide d'une syntaxe illustrée en figure 5.25, afin d'identifier les événements pouvant affecter l'espace (métier ou interactionnel) voisin.

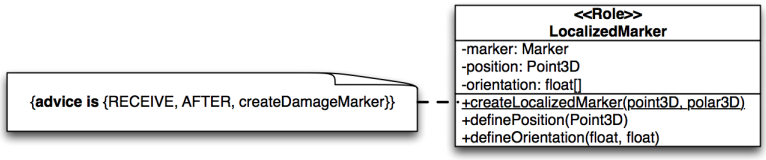


Figure 5.25 – Exemple d'annotation sur une classe « Rôle » issue de l'analyse statique de l'espace métier

Enfin, les Connexions Symphony sont décrites pour chaque événement de connexion. Il s'agit, suivant notre exemple, de décrire la création d'une description de dommage, en réaction à la création d'un Objet Interactionnel Entité « Marqueur » (*Marker*). L'annotation cible le rôle « MarqueurLocalisé » (*LocalizedMarker*), correspondant à une adaptation applicative de l'Objet Métier Entité « Marqueur », plus générique. Afin de préserver les contraintes applicatives de l'espace métier, la création de l'Objet Métier Entité « Dommage » (*Damage*) est déléguée à l'Objet Métier Processus « Réaliser l'état des lieux » (*RealizeInvOfFixtures*).

Les classes de Translation traduisant les événements métier en sémantique d'interaction (et inversement) remplacent finalement les relations « Représente ». Elles sont décrites à l'aide de diagrammes de séquences et, si nécessaire, de scenarii structurés (voir chapitre précédent). La figure 5.26 reprend l'exemple proposé précédemment.

5.8 Synthèse

Nous avons décrit dans ce chapitre la mise en application des principes décrits au chapitre 4, prenant comme illustration le cas d'étude de l'état des lieux, sur lequel nous avons déroulé les phases d'étude préalable, de spécification conceptuelle des besoins, de spécification organisationnelle des besoins, puis la phase d'analyse.

Le déroulement du cycle de développement illustre les activités et collaborations menant au choix de la conception d'une interface en réalité mixte. Nous avons détaillé l'activité de spécification interactionnelle des besoins, particulièrement dense en terme d'artefacts produits et de collaborations entre spécialiste IHM et ergonome. Cette activité recourt ainsi à l'expertise et aux pratiques spécifiques de l'interaction homme-machine pour aboutir à un fragment de solution d'interaction en réalité augmentée répondant aux prescriptions ergonomiques.

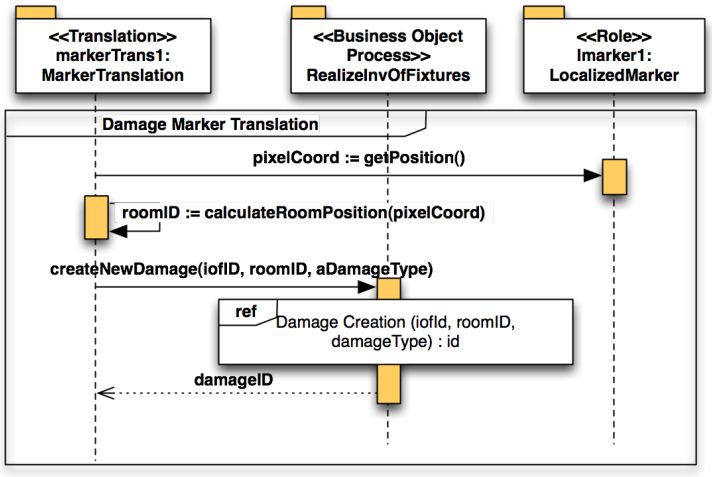


FIGURE 5.26 – Translation entre les rôles d'un marqueur et d'un dommage correspondant à l'événement « créerMarqueurDommage » (*createDamageMarker*)

Par ailleurs, ce chapitre présente le déroulement des activités parallèles de spécification fonctionnelle et de spécification de l'IHM. Les éventuelles incohérences entre les deux spécifications sont traitées par un sous-processus évolutionnel spécifique, permettant de propager les mises en cohérence entre domaines métier et interactionnel de la spécification jusqu'à l'étude préalable.

Nous avons ainsi présenté une illustration quasi exhaustive des différentes activités de la méthode. Comme nous l'avons vu au chapitre précédent, une telle foison de détails ne serait indispensable que dans des conditions de développement particulièrement difficiles, impliquant une équipe cumulant une absence de maîtrise du domaine métier et une quasi absence d'expertise en interaction homme-machine.

*

* *

Dans la partie suivante, nous décrivons l'opérationnalisation du processus, des modèles et langages, ainsi que les évaluations de nos contributions. Cette approche couvre deux types fondamentaux d'outils utilisés dans le cadre d'un développement logiciel, les outils pour la documentation ayant été traité au chapitre précédent : les outils pour l'automatisation de parties du processus et comme supports à l'exécution des applications développées à l'aide de la méthode Symphony.

Troisième partie

Opérationnalisation de la méthodologie : développement applicatif et exécution

CHAPITRE 6

Un intergiciel pour les applications Symphony : Sonata

> when you don't create things, you become defined by your tastes rather than ability. your tastes only narrow & exclude people. so create.
>
> WHY THE LUCKY STIFF AKA _WHY

L'OBJECTIF DE LA phase de conception consiste à intégrer les modèles fonctionnels construits dans la branche gauche de Symphony avec l'architecture technique élaborée dans la branche droite de la méthode. Ces deux aspects sont donc construits en relative indépendance l'un de l'autre et répondent très spécifiquement à des problématiques orthogonales du développement. Cette distinction claire et rigoureuse entre analyse des besoins fonctionnels et techniques constitue l'une de ses forces. Il n'est donc nullement question de proposer dans ce manuscrit une approche prescriptive de la construction du modèle de conception ou des choix techniques à adopter.

Nous proposons néanmoins une approche de la conception des applications développées avec la méthode Symphony étendue, s'appuyant sur un intergiciel spécifiquement adapté, nommé Sonata. Nous illustrons ci-dessous la mise en œuvre de cet intergiciel, en comparant son utilisation à une implémentation classique, basée sur les patrons de conception de l'architecture à composants JavaBeans.

Nos contributions techniques sont focalisées sur les technologies Java™ de la société Sun Microsystems Inc. Cependant, ces technologies (programmation orientée aspect et orientée objets, utilisation de la réflexivité) existant sous une forme ou une autre dans de nombreux langages de programmation, nos contributions sont transposables à d'autres contextes de développement.

6.1 Modèle de conception classique pour les Objets Symphony

Nous présentons dans cette section une approche classique de conception des Objets Symphony, basée sur le modèle à composants JavaBeans. Ce dernier s'appuie sur des conventions de nommage, que nous décrivons brièvement, auxquelles nous ajoutons des conventions d'organisation des classes, spécifiques aux Objets Symphony.

6.1.1 Un modèle à composants basé sur les technologies Java : JavaBeans

Les composants JavaBeans constituent l'une des technologies d'intégration logicielle basée sur la technologie Java. Les JavaBeans correspondent à des classes Java encapsulées dans des fichiers d'archive `.jar`. Une large part de la technologie JavaBeans est basée sur des conventions de nommage des différents éléments contenus dans les classes Java, permettant aux mécanismes de gestion de prendre en charge les composants JavaBeans de façon optimale. Concernant les attributs des classes JavaBeans, ces conventions sont les suivantes :

- l'ensemble des attributs de la classe doivent être déclarés comme privés (c'est-à-dire, non modifiables depuis l'extérieur de la classe) ;
- l'accès en lecture à l'attribut doit être assuré par une méthode dont le nom correspond à celui de l'attribut préfixé de `get` ;
- l'accès en écriture doit être assuré par une méthode dont le nom correspond à celui de l'attribut préfixé de `set`.

Par exemple, l'attribut `size`, déclaré par l'expression `private int size` et dénotant la taille d'un composant JavaBeans graphique, peut être lu grâce à l'appel à la méthode `int getSize()` et modifié grâce à l'appel à la méthode `void setSize(int size)`.

La technologie JavaBeans s'appuie également sur le patron de conception Observateur/Observable décrit par Gamma et al. [52]. Ce dernier consiste à proposer à des composants logiciels d'observer les modifications pouvant intervenir sur les propriétés d'autres composants, sur la base d'un « contrat d'observation » générique. Lors d'un tel événement, le composant observé notifie donc l'ensemble de ses observateurs. La répercussion de changements sur les composants via le contrat générique maintient donc le découplage entre ceux-ci et permet à de nouveaux composants, non anticipés par le concepteur, de s'intégrer dans des applications existantes.

Les JavaBeans exploitent un modèle d'observation plus spécifique : un certain nombre d'événements est prédéfini, auxquels correspondent des observateurs (*listeners* en termes Java). Un composant souhaitant être notifié d'un certain type d'événement doit solliciter le service du composant observé dont le nom correspond au *listener*, préfixé de `add`. Dans le cadre des composants graphiques Java, il existe par exemple un événement `MouseMotion`, déclenché lorsque le pointeur se déplace dans la zone occupée par le composant, ainsi qu'un observateur `MouseMotionListener` reconnu par les composants déclenchant cet événement. Afin d'être notifié des mouvements du pointeur, un composant JavaBeans doit donc appeler la méthode `addMouseMotionListener` auprès du composant qu'il souhaite observer. Les notifications prennent la forme d'appels de méthode `mouseMoved`, que doivent implémenter les observateurs. La figure 6.1 propose un exemple simplifié d'application du patron Observateur/Observable, dans le contexte des JavaBeans.

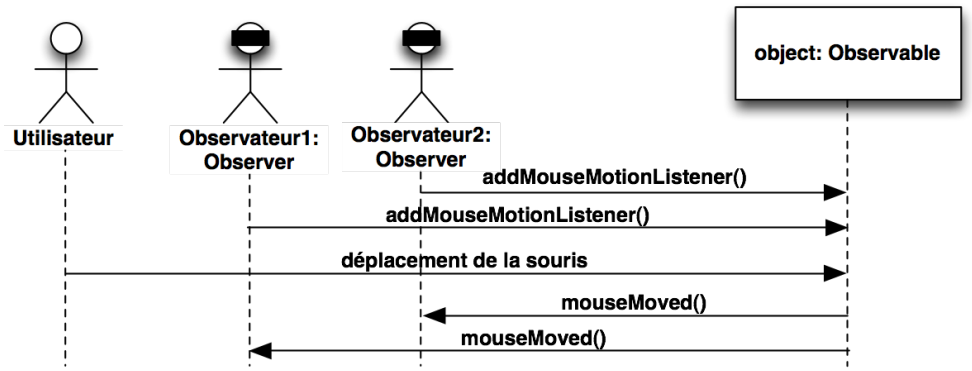


Figure 6.1 – Exemple d'application du patron Observateur/Observable, dans le contexte des JavaBeans

Le modèle JavaBeans propose également un modèle de persistence basé sur l'utilisation de l'interface de marquage `Serializable` ainsi que sur des propriétés appliquées aux attributs des composants. Les interfaces de marquage sont des classes abstraites vides – elles ne déclarent ni méthodes ni attributs –, dont la fonction est de permettre la classification des classes les implémentant. L'interface `Serializable` déclare ainsi auprès de la machine virtuelle Java que l'état des instances des classes l'implémentant sont susceptibles d'être sauvegardées sous la forme de fichiers binaires.

L'ensemble des propriétés décrites ci-dessus permet notamment, grâce aux mécanismes d'introspection intégrés au langage Java, d'assembler et de configurer les composants JavaBeans depuis des environnements graphiques de développement.

6.1.2 Modèle de conception Symphony adapté à la technologie JavaBeans

Dans le cadre de ces travaux sur la méthode Symphony originelle [63], E. Jausseran propose deux constructions systématiques des Objets Symphony de niveau conception, vers les modèles à composants JavaBeans et Entreprise JavaBeans (EJB). Nous traitons ici la première construction, illustrée pour les Objets Entité par la figure 6.2.

Trois points essentiels définissent la construction des Objets Symphony de niveau conception sous la forme de composants JavaBeans :

1. L'instanciation et la configuration des Objets Symphony sont assurées par un objet (`ObjectAFactory`) unique, assumant les rôles de « fabrique » et de « singleton », au sens de Gamma et al. [52]. Les fabriques ont également pour responsabilité d'affecter un identifiant unique à chaque Objet Symphony,
2. Les objets stéréotypés « Rôle » sont les délégués des Objets Symphony avec lesquels `ObjetA` collabore. Cette approche permet d'encapsuler le déroulement de la collaboration entre Objets Symphony dans des classes bien définies. Par conséquent, dans l'hypothèse d'une évolution de l'organisation des Objets Symphony, seules les classes rôles sont impactées, limitant ainsi la portée des modifications du code d'implémentation. L'intégralité de l'interface des Objets Symphony cibles est répliquée dans les classes rôles correspondantes. La sémantique de ces méthodes peut

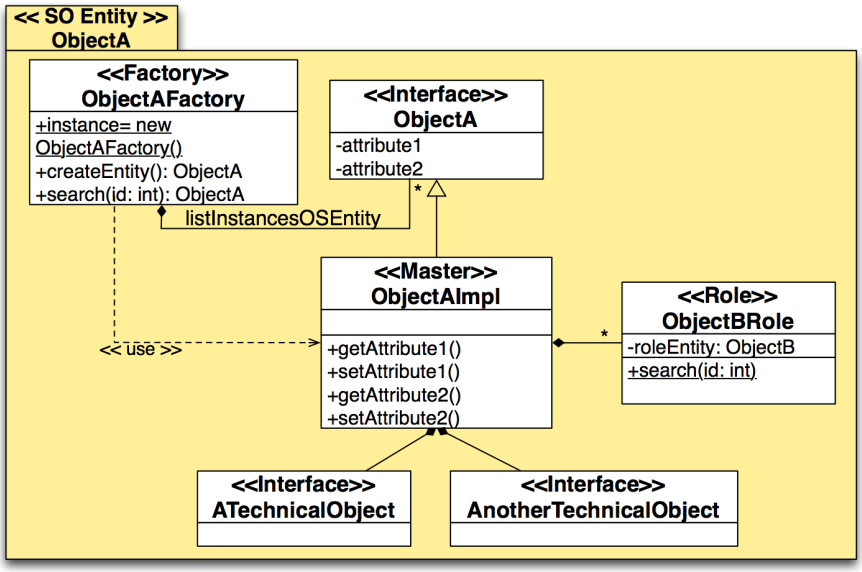


FIGURE 6.2 – Objet Symphony Entité de niveau conception adapté à l'architecture JavaBeans, d'après E. Jausseran [63]

être modifiée ou bien de nouvelles méthodes peuvent les compléter, en fonction du contrat liant les Objets Symphony source et cible.

3. Le code technique (tel que le code faisant appel à des bibliothèques graphiques, à la couche de persistance…) n'est pas utilisé directement par les Objets Symphony, mais est encapsulé dans des composants de type JavaBeans. Des classes interface (dans la figure 6.2, `ATechnicalObject` et `AnotherTechnicalObject`) permettent de préciser les services attendus par l'Objet Symphony et que ces composants devront implémenter. Cette approche permet de découpler les composants supposés stables (c'est-à-dire, les Objets Symphony) des portions du code d'implémentation ayant une plus large tendance à évoluer (c'est-à-dire, le code technique).

Ce modèle de conception s'est avéré être tout autant efficace que simple à développer dans le cadre des applications conçues avec la méthode Symphony originelle. Toutefois, plusieurs points entament sa pertinence lorsque utilisé pour les applications issues de notre méthode Symphony étendue :

1. Le comportement des objets fabriques (dans notre exemple, `ObjectAFactory`) ne varie que très peu d'un Objet Symphony à l'autre. Concrètement, seules les méthodes liées à la configuration des instances nouvellement créées distinguent ces objets fabriques les uns des autres. L'implémentation des classes fabriques implique donc un grand nombre de réplications de blocs de code,
2. Le mécanisme JavaBeans d'observation n'est valable que pour les événements prévus par les bibliothèques techniques Java. La prise en compte d'autres événements, telle la création d'un marqueur dans notre exemple de gestion des états des lieux, doit être anticipée dans les composants susceptibles d'être observés. En termes du langage Java, cela se concrétise par l'implémentation d'une interface (`Observable`) et l'appel à une méthode (`notifyObservers`) pour chaque événement susceptible d'être pertinent pour un composant tiers. Dans une optique de conception pour la réutilisabilité,

il est donc nécessaire de multiplier les appels à la méthode `notifyObservers`, au détriment de la lisibilité du code d'implémentation.

3. Ce modèle ne propose pas de solution quant à l'implémentation des Translations [1] (voir section 4.3.2). Par défaut, d'aucuns tendent à appeler les instances de Translation depuis le code d'implémentation des classes « Maître » des Objets Symphony, avec pour conséquence fâcheuse d'augmenter conséquemment le couplage [2] entre objet source, objet(s) cible(s) et Translation.

Dans les paragraphes suivants, nous présentons l'intergiciel Sonata, dédié aux applications conçues à l'aide de la méthode Symphony étendue et traitant, entre autres, les trois points ci-dessus.

6.2 Un intergiciel orienté services : Sonata

L'intergiciel Sonata a été conçu pour répondre à deux besoins, tout d'abord celui de faciliter l'implémentation et l'exécution des applications structurées à base d'Objets Symphony, tout en minimisant les efforts de configuration et la réplication du code d'implémentation. Le second besoin, plus technique, est de minimiser le couplage, c'est-à-dire les dépendances fonctionnelles, entre Objets Métier et Objets Interactionnels. Le couplage se manifeste, entre autres, par des appels à des méthodes externes à un composant. Il s'agit donc de minimiser les appels de méthodes en Objets Métier et Objets Interactionnels. En effet, nous estimons qu'inclure un code du domaine métier dans un composant de l'interaction, ou inversement un code lié à l'interaction homme-machine dans un composant du domaine métier, nuit gravement à la réutilisabilité des composants concernés.

Nous souhaitons également proposer une alternative au modèle d'observation des JavaBeans, pour les événements non pris en compte par les bibliothèques techniques.

Nous présentons dans cette section les concepts sous-tendant l'intergiciel. Des transformations de modèles, facilitant la mise en œuvre d'applications utilisant Sonata, sont décrites dans le prochain chapitre. La structure de l'intergiciel est, elle, décrite en annexe A.

Considérant le fait que nos contributions se réfèrent fortement aux principes de la programmation orientée aspects (POA, en anglais *aspect-oriented programming* ou AOP), nous en décrivons les principes à la sous-section suivante avant d'aborder l'intergiciel Sonata proprement dit.

6.2.1 Principes de la conception orientée aspects

La programmation par aspects [68], dont les principes ont été décrits par Kiczales et al., est un paradigme de conception logicielle préconisant la séparation des préoccupations, c'est-à-dire la distinction entre implémentation fonctionnelle (liée au domaine métier, notamment) et la prise en compte des besoins non fonctionnels tels que la journalisation, la sécurité ou la persistence.

1. Les Translations sont les classes dans lesquelles est décrite la sémantique des connexions entre Objets Métier et Objets Symphony.
2. Nous assumerons, dans l'attente d'une définition plus précise, que le couplage quantifie le niveau de dépendance fonctionnelle entre entités logicielles.

L'approche concrète de la programmation orientée aspects consiste à identifier des *points de jonction* (ou *join points*) dans le logiciel – en préliminaire à l'appel d'une méthode, à la suite d'un appel ou bien en substitution à un appel de méthode –, dans lesquels seront *tissés* – c'est-à-dire, insérés dans le code binaire – des *greffons* (ou *advice*) – un code d'implémentation traitant la préoccupation de l'aspect.

La POA est une technologie générique, pouvant être adaptée à la plupart des langages orientés objet ou fonctionnels. La bibliothèque AspectJ [ASP] pour la langage Java est l'un des modules de POA les plus utilisés.

6.2.2 Modèle de conception adapté à l'intergiciel Sonata

Sonata permet de connecter aisément les « briques » logiques des applications développées à l'aide de la méthode Symphony, c'est-à-dire les composants techniques, les Objets Symphony et les Translations, tout en facilitant leur réutilisation. Le code d'implémentation de l'intergiciel est disponible en ligne à l'adresse :

```
http://github.com/ggodet-bar/Sonata/tree/master
```

Afin de réaliser cet objectif, Sonata fixe plusieurs aspects de la construction du modèle de conception des Objets Symphony. Ces évolutions s'appuient pour l'essentiel sur le principe d'utilisation de conventions, concernant la structuration, le nommage ou l'organisation des composants, en lieu et place des traditionnelles configurations au cas par cas. L'élaboration de solutions de conception perd ainsi une certaine souplesse mais gagne en lisibilité et efficience. Cette approche s'applique naturellement aux Objets Symphony, dont la structure interne et l'organisation sont systématiques.

Nous basons la description des mécanismes de Sonata sur le modèle de conception classique décrit ci-dessus, dont nous décrivons dans les paragraphes suivants les différentes évolutions.

6.2.2.1 Organisation initiale des Objets Symphony en phase de conception

Considérant l'utilisation de Sonata, le modèle de conception des Objets Symphony se limite à celui présenté en figure 6.3, similaire au modèle d'analyse. Cependant, afin d'assurer l'intégration dans l'intergiciel, les conventions suivantes doivent être respectées :

- les services proposées par l'Objet Symphony doivent être décrits dans une classe interface ;
- l'implémentation de ces services doit être décrite dans une classe dite « Maître » ; le nom de la classe maître doit correspondre au nom de la classe interface, suivie du suffixe `Impl` (étant donné une classe d'interface `ObjectA`, la classe maître doit être nommée `ObjectAImpl`) ;
- chaque relation (d'utilisation ou de représentation) entre deux Objets Symphony doit être matérialisée par une classe rôle liée à la classe maître ;
- l'ensemble des classes d'un même Objet Symphony doivent être intégrées dans le même paquetage Java ; ce paquetage doit avoir le même nom que la classe interface (tout en respectant les conventions Java de nommage des paquetages ; reprenant notre exemple, le paquetage doit être suffixé par `objecta`).

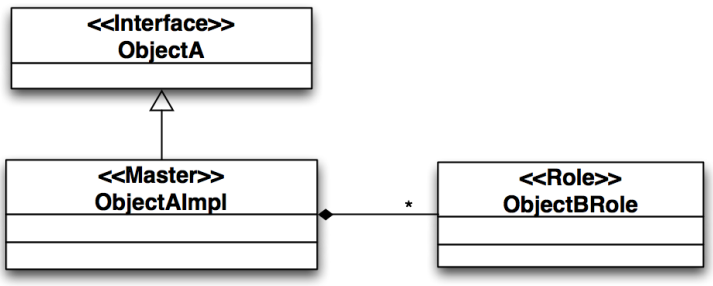


Figure 6.3 – Exemple d'organisation initiale d'Objet Symphony, en phase de conception

6.2.2.2 Utilisation d'interfaces de marquage

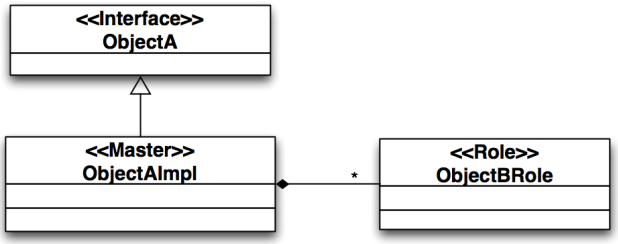
Sonata reconnaît deux types d'interfaces de marquage : d'une part `EntityObject` et `ProcessObject`, d'autre part `SymphonyRole`. Les premières permettent d'identifier les classes « Maître » des Objets Symphony Entité et des Objets Symphony Processus. Elles permettent à l'intergiciel d'enregistrer les Objets Symphony ainsi identifiés auprès de la fabrique abstraite `AbstractEntityFactory` (c.f section 6.2.2.3). À l'instar des premières, l'interface `SymphonyRole` permet d'identifier les classes « Rôle » d'un Objet Symphony auprès de l'intergiciel.

Les interfaces de marquage reconnues par Sonata permettent également à l'intergiciel de tisser des services récurrents au sein des classes des Objets Symphony. Ce mécanisme est décrit en figure 6.4, dans le cas de la conception d'un Objet Symphony Entité. Le fait d'implémenter l'interface `EntityObject` permet à Sonata de tisser dans l'objet « Maître » des services d'identification (deux méthodes, `void setID(int id)` et `int getID()`, encapsulant un attribut privé int id). De même, le fait d'implémenter l'interface `SymphonyRole` permet à Sonata de tisser dans chaque objet « Rôle » des services identifiant les instances d'Objets Symphony collaborateurs. Ces différents services sont accessibles via des appels de méthode du type :

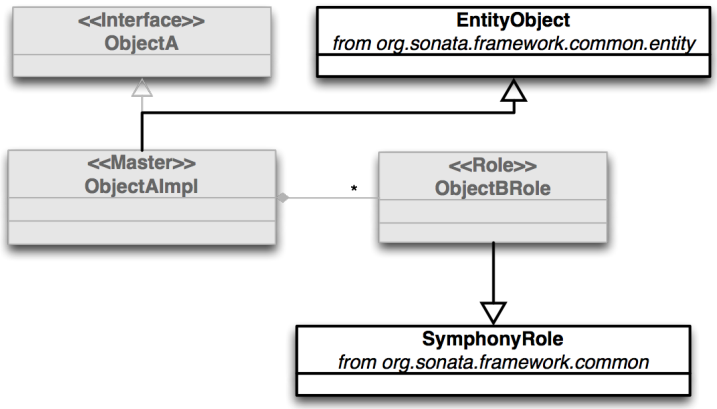
```
Dommage unDommage = // Code d'initialisation du dommage
int unIdentifiant = ((EntityObjectServices)unDommage).getID() ;
// Utilisation de l'identifiant du dommage
```

6.2.2.3 Factorisation des classes fabriques

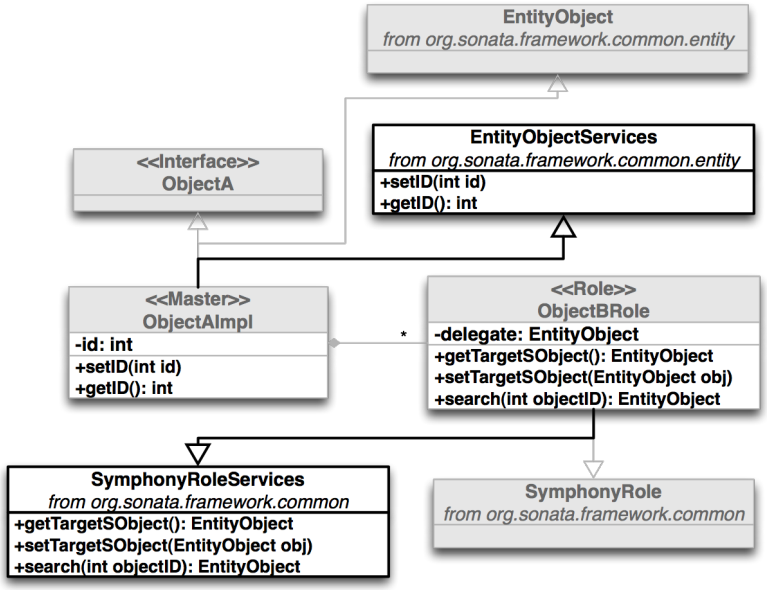
Les objets fabriques remplissent deux rôles : 1) la gestion des instances d'Objets Symphony dont ils ont la responsabilité et 2) la configuration de ces instances. La gestion des instances consiste en un ensemble de responsabilités récurrentes d'un Objet Symphony à l'autre : l'instanciation d'Objets Symphony, la recherche d'objets à partir de leur identifiant et la suppression d'un objet. La configuration des instances implique d'une part l'affectation de propriétés définies dans des fichiers de configuration (par exemple, la taille d'affichage souhaitée d'une fenêtre, en pixels), d'autre part l'instanciation des classes techniques (par exemple, les classes de gestion de la persistance ou de l'affichage) utilisées par les Objets Symphony, dont le choix est également effectué au travers de fichiers de configuration – les détails de ce mécanisme sont abordés dans la sous-section suivante.



(a) Structure initiale de l'OS ObjectA en phase d'Analyse



(b) Ajout des interfaces de marquage par le développeur en phase de Conception



(c) Tissage automatique de services effectué par Sonata, à l'exécution

FIGURE 6.4 – Étapes du tissage de services au sein des Objets Symphony

L'implémentation des classes fabriques étant largement répétitive, nous en avons déplacé et factorisé les responsabilités dans l'intergiciel Sonata : une seule fabrique générique `AbstractEntityFactory`, représentée en figure 6.5 gère l'ensemble des instances d'Objets Symphony créées au cours du cycle de vie de l'application.

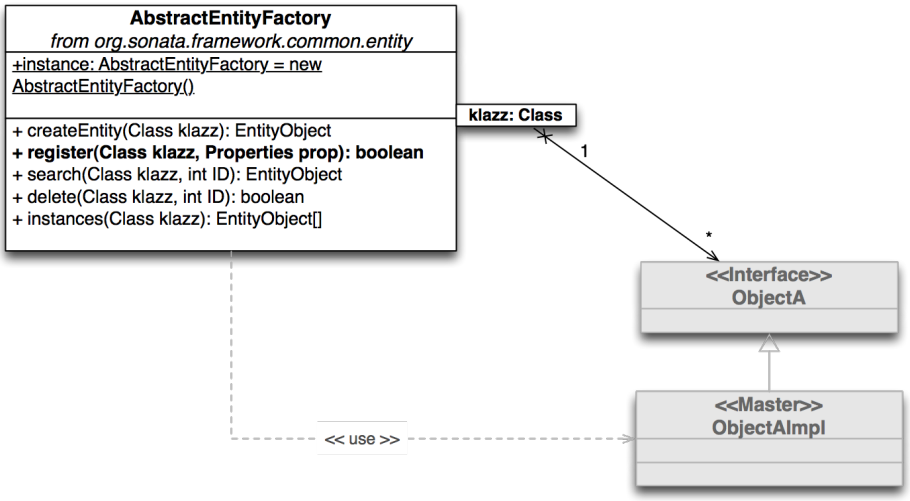


FIGURE 6.5 – Fabrique abstraite de l'intergiciel Sonata

L'instance de `AbstractEntityFactory` gère l'ensemble des types d'Objets Symphony s'étant enregistrés auprès de celle-ci grâce à l'appel à la méthode `register`, effectué généralement lors de l'initialisation de l'application. Par la suite, les instances d'Objets Symphony sont crées via un appel à une méthode `createEntity`. L'exemple ci-dessous décrit l'instanciation d'un Objet Métier Entité Dommage, à l'aide de l'entité `AbstractEntityFactory`.

```
Dommage unDommage = AbstractEntityFactory.instance.createEntity
    (Dommage.class) ;
```

Quant à la configuration des instances d'Objets Symphony, celle-ci s' effectue sur la base des valeurs définies dans les fichiers de configuration. L'affectation d'une valeur à l'attribut correspondant s'appuie sur la même règle que celle définie pour l'accès aux attributs privés des composants JavaBeans. Ainsi, une donnée de configuration dont le nom est `size` et la valeur `140` générera un appel de la forme `setSize(140)`.

6.2.2.4 Inversion de dépendance des classes techniques

Le principe de l'inversion de dépendance, décrit par R. C. Martin [75], consiste à contourner certaines situations d'instabilité récurrentes dans les systèmes logiciels, lorsque des classes dites abstraites (au sens où ces classes définissent des comportements à haut niveau d'abstraction) référencent des classes dites concrètes (le plus souvent, des classes intégrant du code technique). La solution la plus immédiate consiste à limiter les appels de la classe abstraite à des requêtes auprès d'interfaces, que devront implémenter les classes techniques. Ces dernières sont alors dépendantes de ces interfaces, d'où le terme d'inversion de dépendance. Cette approche est décrite en figure 6.6.

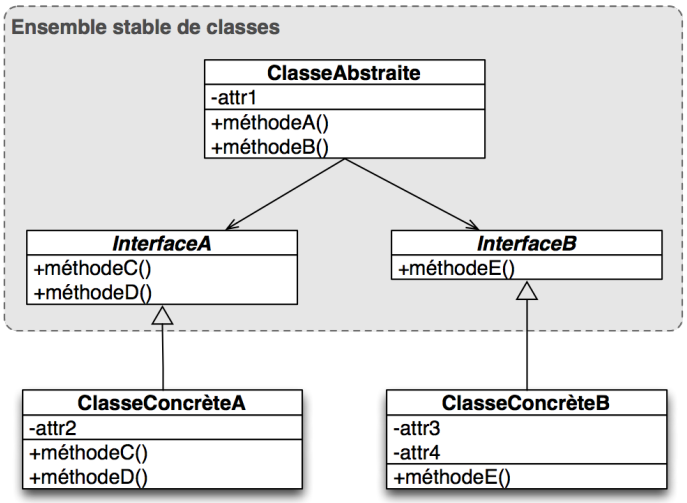


FIGURE 6.6 – Exemple type d'inversion de dépendance, d'après R. C. Martin [75]

Le point de variabilité de cette solution réside dans la façon d'instancier les classes techniques et de les affecter aux classes abstraites. Afin de maximiser l'autonomie des Objets Symphony vis-à-vis de leur « couche technique », l'intergiciel Sonata assume cette responsabilité. Le choix des classes techniques est ainsi réalisé lors de l'intégration des différents composants du système : celles-ci sont identifiées dans les fichiers de configuration.

Lors de l'exécution de l'application, l'instance de fabrique abstraite est configurée pour réaliser l'appariement idoine entre Objets Symphony et composants techniques, lors de l'instanciation des premiers : la fabrique abstraite charge les classes correspondant aux composants techniques utilisés par chaque Objet Symphony lors de l'initialisation du système. Lors de l'instanciation d'un Objet Symphony, la fabrique abstraite créée une nouvelle instance des composants techniques nécessaires et les associe aux Objets Symphony.

6.2.2.5 Externalisation des connexions

Les modèles d'analyse distinguent nettement les fonctionnalités liées au domaine métier de celles liées à l'interaction, grâce à l'organisation de l'application sous la forme d'Objets Métier et d'Objets Interactionnels. De même, le comportement définissant la connexion entre Objets Métier et Objets Interactionnels est décrit de manière indépendante du reste de l'application, grâce aux Translations. L'approche de l'intergiciel Sonata est de maintenir ces distinctions jusque dans le code d'implémentation. Considérant que les connexions doivent être intégrées transversalement au reste du code fonctionnel, l'approche de la programmation orientée aspects semble adaptée.

Dans la section 5.7, nous avons présenté un mécanisme d'annotation permettant d'associer la méthode d'une classe Translation à un appel de méthode déclenché dans l'espace métier ou interaction. Nous recourons à cette formalisation pour générer automatiquement, grâce aux outils de développement supportant Sonata, les aspects permettant de lier Objets Symphony et Translation.

Nous présentons ci-dessous, à titre indicatif, l'aspect généré depuis l'annotation présentée en figure 5.25.

```
1  package control.communication;
2  //  Importations
3
4  public aspect MarkerRoleAspect {
5
6    pointcut MarkerRoleCalls(): execution(public * MarkerRole
         .*(..)) ;
7
8    after() returning(MarkerRole target): execution(MarkerRole
         createMarkerRole(Point3D, Point3D)) {
9      try{
10       Invoker.instance.createRequest((SymphonyObject)target.
             getTargetSObject(), "createDamageMarker");
11       Invoker.instance.sendRequest();
12     } catch (Exception e) {/* Gestion des exceptions */}
13   }
14 //  Autres connexions
15 }
```

Bien que d'un abord relativement complexe, ces classes, générées automatiquement, n'ont pas vocation à être modifiées par les développeurs et peuvent donc être ignorées par ces derniers.

La première expression de l'aspect (l. 6) restreint le champ d'application de l'aspect aux seuls appels effectués sur la classe `MarkerRole`. Les lignes suivantes (l. 8 – l. 13) correspondent à la traduction de l'annotation proprement dite. Les points saillants de l'exemple sont les lignes 10 et 11, qui laissent apparaître deux appels au mécanisme central de Sonata : l'instance de la classe `Invoker`. L'`Invoker` a pour responsabilité de gérer la répercussion des appels depuis l'aspect où ils sont capturés vers l'instance idoine de Translation. La structure du mécanisme sur lequel s'appuie l'Invoker est exposée en annexe A.

Nous présentons en figure 6.7 différentes étapes du tissage d'aspect avec un Objet Symphony déclencheur d'*événements de connexion*. La partie supérieure de la figure 6.7 présente une vue simplifiée de la séquence correspondant à la création d'un marqueur, déjà décrite au chapitre 5. La partie centrale de la figure 6.7 montre le résultat du tissage de l'aspect présenté plus haut, correspondant à un appel aux méthodes `createRequest` et `sendRequest`. Lors de la résolution de la requête, un Objet Métier Entité Dommage est créé, à l'initiative de l'instance de la Translation. Aussitôt créée, l'instance de l'OME Dommage est liée à celle de l'OIE Marqueur (appel à la méthode `bind`). Par souci d'exhaustivité, la partie inférieure de la figure 6.7 reprend la résolution de l'appel à la méthode `createMarkerDamage()` également présentée au chapitre 5.

Comparé au système de notification proposé par le modèle JavaBeans, l'utilisation conjointe de Sonata et de la génération d'aspects permet de retirer de la logique métier ou interactionnelle la problématique de communication avec d'autres composants, qui est alors gérée de façon transversale et indépendante.

6.2.3 Configuration de l'intergiciel

Plusieurs mécanismes permettent d'adapter le comportement de Sonata aux besoins des concepteurs :

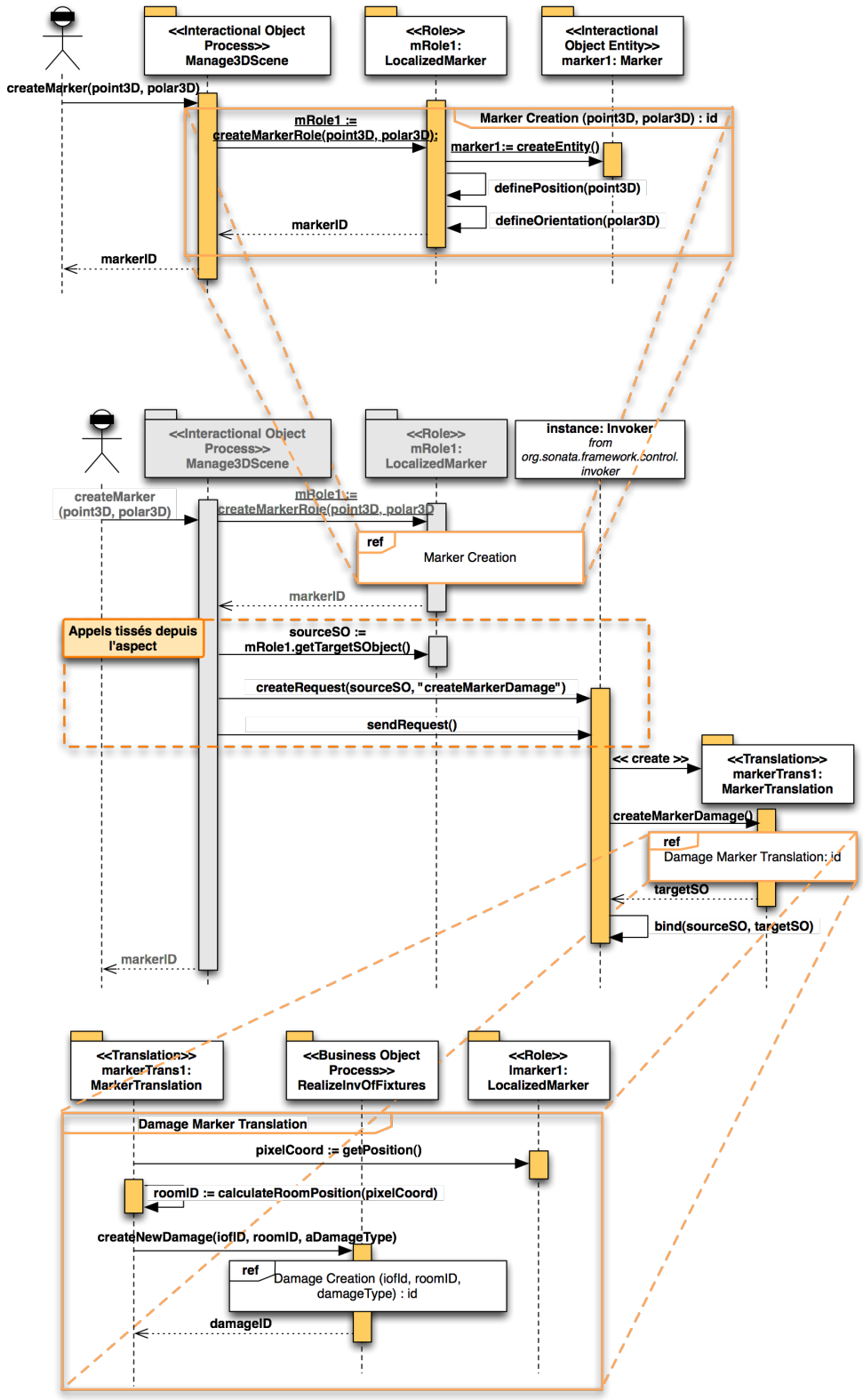


FIGURE 6.7 – Exemple de résolution simplifiée du tissage d'une connexion dans l'espace interaction

- un paramétrage de l'Invoker permet d'ignorer l'Objet Processus lors de la résolution des événements de connexion, permettant ainsi de tester le comportement des connexions Symphony (objet source, objet(s) cible(s) et Translation) indépendamment du reste de l'application ;
- les conventions par défaut de l'intergiciel peuvent être contournées en redéfinissant explicitement les classes constituant chaque Objet Symphony ;
- des valeurs par défaut peuvent être définies pour chaque Objet Symphony ; celles-ci sont affectées à chaque instance lors de sa construction par l'`AbstractEntityFactory`.

Nous présentons dans la section suivante quelques utilisations de Sonata pour des application exposant une interaction classique ou en réalité mixte.

6.3 Applications de l'intergiciel Sonata

Au cours de nos travaux, nous avons développé trois applications dont la fonction, outre de servir de cas d'étude pour la spécification de la méthode Symphony étendue, était d'estimer la pertinence de l'intergiciel Sonata pour des applications différentes du point de vue des types d'interaction et des technologies utilisées.

L'application d'état des lieux augmenté, dont nous avons utilisé le développement comme cas d'étude de ce manuscrit, s'appuie sur un domaine métier de faible complexité (4 Objets Métier) et un domaine interactionnel riche et complexe (6 Objets Interactionnels), utilisant une interface de type OpenGL [Ope] distribuée sur des lunettes de vision augmentée et un smartphone.

En complément à l'application d'état des lieux augmenté, nous avons envisagé le développement d'une application suivant un cahier des charges plus complexe. À ce titre, nous avons collaboré avec l'équipe VASCO [3] du Laboratoire d'Informatique de Grenoble dans le cadre d'un projet d'évaluation de la sécurité aéroportuaire, nommé EDEMOI [4]. L'application développée dans le cadre de cette collaboration a pour fonction de traduire les résultats de test, délivrés par les outils proposés par VASCO sous forme de tableaux peu lisibles, en animations graphiques. L'analyse et la réorganisation des tests en animations constituent la partie métier du projet (5 Objets Métier). Le rendu graphique mêle rendu classique utilisant la bibliothèque Swing [Swi] du langage Java, ainsi que la bibliothèque de manipulation d'images vectorielles Batik [Bat]. Actuellement cantonnée aux stations de travail, l'interface de cette application doit être transposée sur table augmentée [11], avec pour objectif de réutiliser les six Objets Interactionnels déjà implémentés et de ne modifier que les composants techniques du système.

La troisième application, nommée TeamAvatars, a été développée suivant des contraintes de simplicité et de réutilisation. Celle-ci doit ainsi servir d'application « type » démontrant les fonctionnalités de l'intergiciel en un nombre minimal de lignes de code d'implémentation, indépendamment de la problématique de construction des systèmes de réalité mixte. TeamAvatars est composée de trois Objets Métier et de trois Objets Interactionnels. Les deux Objets Interactionnels Entités de TeamAvatars sont issus de l'application EDEMOI ; par conséquent seul l'Objet Interactionnel Processus est propre à l'application.

3. `http://vasco.imag.fr`
4. Élaboration d'une DÉmarche et d'outils pour la MOdélisation Informatique, la validation et la restructuration de réglementations de « sûreté » (sécurité), et la détection des biais dans les aéroports

Enfin, signalons que nous avons également expérimenté, avec succès et un effort minimal, l'implémentation des Objets Symphony, instrumentés avec Sonata, dans l'intergiciel à composants OSGi [OSG].

6.4 Synthèse

Nous avons décrit dans ce chapitre l'une de nos contributions techniques : l'intergiciel Sonata, dont les fonctions sont de faciliter l'implémentation des applications développées à l'aide de la méthode Symphony et de minimiser le couplage entre Objets Symphony.

Concernant le premier point, Sonata combine plusieurs approches : l'exploitation des conventions d'organisation et de structuration des Objets Symphony, afin de minimiser les efforts de configuration, la factorisation des comportements redondants, et l'automatisation de la configuration des composants. Une large partie de ces approches s'appuie sur les principes de la programmation orientée aspects ; toutefois, les aspects sont soit intégrés à l'intergiciel (par exemple, dans le cas du tissage des classes `EntityObjectServices` et `SymphonyRoleServices`), soit générés automatiquement (dans le cas du tissage de connexion entre Objets Symphony et Translations). Nous espérons ainsi soulager le concepteur de la manipulation des aspects.

Quant au second point, nous le traitons grâce à l'externalisation des connexions Symphony. Ainsi, outre la focalisation du couplage entre Objets Symphony dans les classes « Rôle », le couplage entre Objets Métier et Objets Interactionnels est inexistant. Plus exactement, ce dernier type de couplage est concentré dans les classes Translation, dont la vocation est d'être instables et propres à l'application.

Des applications aux interfaces et complexités diverses ont été implémentées et font appel à l'intergiciel Sonata, confirmant empiriquement l'effectivité de notre contribution. Une évaluation plus approfondie est exposée au chapitre 8.

*

* *

Le chapitre suivant expose les automatisations instrumentant les modèles de la méthode Symphony étendue, de la spécification jusqu'à l'implémentation. La génération des aspects systématiques de la configuration de Sonata sont également abordés.

CHAPITRE 7

Transformations de modèles pour le raffinement, la cohérence, la traçabilité et la génération

> L'intelligence est caractérisée par la puissance indéfinie de décomposer selon n'importe quelle loi et de recomposer en n'importe quel système.
>
> *L'évolution créatrice*
> HENRI BERGSON

SIMPLIFICATION des activités de conception et performances à l'exécution ne sont pas les seuls apports nécessaires pour justifier l'utilisation de la méthode Symphony étendue. En l'état, de nombreuses activités systématiques seraient à la charge du concepteur, lequel serait alors noyé dans le flot de construction et de raffinement de produits, accumulation ingérable et *de facto* d'informations à traiter et maintenir.

Comment envisager, dans ce contexte, de tracer les besoins utilisateurs de leur expression jusqu'à l'implémentation ? Comment faciliter la mise en cohérence de produits construits par différents acteurs de la conception ? Nous traitons cette problématique en définissant un processus de raffinement de modèles, destiné à la gestion des différents modèles d'Objets Symphony d'une part et à la gestion de leur cohérence et de leur traçabilité d'autre part.

Ce chapitre introduit nos objectifs d'automatisation, puis nous détaillons les différents méta-modèles formalisant le domaine conceptuel des Objets Symphony. Cette formalisation donne lieu à plusieurs types de transformations, dont l'objectif est de faciliter l'opérationnalisation de la méthode et d'assurer la cohérence de ses produits.

7.1 Objectifs d'automatisation et profilage UML

En matière d'automatisation du processus Symphony, nous focalisons nos contributions sur les activités de raffinement des Objets Symphony. Leur niveau de formalisation facilite en effet leur intégration dans les outils de transformation de modèles.

Deux types d'opérations sont appliquées aux Objets Symphony au cours de la conception :

- les transformations, dont l'objectif est d'automatiser une étape de création ou de modification de produit ;
- les vérifications de cohérence, dont l'objectif est de valider le respect de contraintes, pour des modèles existants.

Les transformations sont appliquées pour :

- rassembler les modèles partiels de niveau spécification organisationnelle et interactionnelle des besoins (modèle des Objets Métier, modèle des Objets Interactionnels et modèle des connexions Symphony entre Objets Métier et Objets Interactionnels) ;
- raffiner le modèle global de niveau spécification organisationnelle et interactionnelle des besoins en modèle de niveau analyse ;
- raffiner le modèle de niveau analyse en modèle de niveau conception, utilisant l'intergiciel Sonata ;
- raffiner le modèle de niveau conception en modèle d'implémentation (c'est-à-dire, le code d'implémentation).

Les vérifications de cohérence peuvent être déclinées en deux types distincts :

- la vérification de la conformité d'un modèle par rapport à son métamodèle, généralement réalisée automatiquement par la plupart des outils de modélisation ;
- la vérification de la cohérence entre deux modèles, qu'ils soient instance du même métamodèle ou de métamodèles différents.

Les vérifications de cohérence entre modèles s'apparentent au tissage de modèles [43], non au sens de la programmation orientée aspects, mais comme un moyen d'établir des correspondances entre les éléments de différents modèles et de maintenir ces correspondances tout au long de leur cycle de vie. Ces opérations sont toutes indiquées pour le maintien de la traçabilité des produits, que nous appliquons aux Objets Symphony.

Les Objets Symphony constituent une utilisation spécifique du langage UML : par exemple, les modèles de niveau spécification organisationnelle et interactionnelle des besoins sont des diagrammes de classes, limités à des paquetages reliés entre eux via des relations de dépendance spécifiques.

Or, le langage UML peut être étendu et adapté à des domaines variés, à l'aide de profils. Les profils UML constituent un mécanisme d'extension de métamodèles existants, dans une optique d'adaptation. Les plus communs concernent des plate-formes logicielles, telles que J2EE [J2E] et .NET [DOT], ainsi que des domaines d'application, tels que les systèmes temps-réel ou la modélisation des processus métier [89].

Nous présentons en figure 7.1 un extrait de la superstructure UML [89], ciblée sur les concepts manipulés dans ce chapitre, et fortement simplifiée, par souci d'en conserver la lisibilité. En particulier, plusieurs relations transitives d'héritages sont simplifiées en un héritage simple. De même, certains attributs présents dans un élément sont originellement dérivés de classes

héritées.

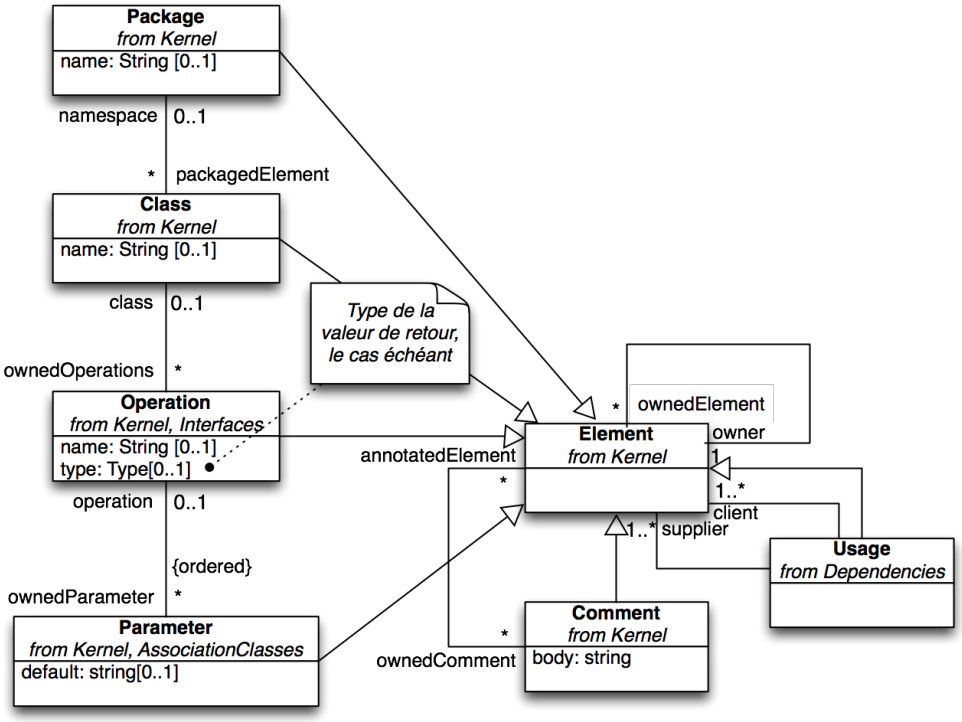


Figure 7.1 – Extrait et simplification de la superstructure UML [89]

La superstructure UML correspond au métamodèle du langage, lequel décrit une architecture d'éléments conceptuels, indépendamment des différents diagrammes dont ces derniers constituent différentes représentations. L'ensemble des diagrammes et représentations des concepts UML constitue d'ailleurs l'infrastructure UML [86]. À titre d'exemple, les diagrammes de séquences et les diagrammes d'interaction exploitent les mêmes composants conceptuels mais utilisent des représentations différentes.

La standardisation du mécanisme d'extension du langage UML facilite la prise en charge des profils par les outils de modélisation existants, et par conséquent l'intégration de modèles spécifiques à une organisation dans des flots de modélisation, transformation et génération. Nous envisageons donc de construire plusieurs profils UML, chacun dédié à un niveau de modélisation (spécification ou analyse) des Objets Symphony.

Nous décrivons dans la section suivante les différents métamodèles conceptuels représentant les modèles Symphony, puis leur incorporation aux profils UML, avant de décrire les transformations et vérifications de cohérence élaborées dans le contexte de nos travaux.

7.2 Métamodèles des Objets Symphony de niveau spécification et analyse

Les Objets Symphony sont l'un des deux modèles consensuels sur lesquels s'articule la méthode Symphony étendue. L'opérationnalisation et la cohérence des différents modèles et transformations intégrés dans le processus de raffinement des Objets Symphony, de

la spécification jusqu'à l'implémentation, sont donc essentielles. Nous envisageons bien évidemment une automatisation partielle, qui ne prétend pas se substituer à l'expertise concepteur.

Les composants conceptuels que nous employons dans ce manuscrit sont généralement représentés dans l'infrastructure UML sous la forme de diagrammes de classes. Il s'agit des concepts de paquetage, classe, opération (c'est-à-dire, de méthode), paramètre, commentaire, et la relation de dépendance « utilise » (*Usage*, sur la figure 7.1).

Dans les paragraphes suivants, nous décrivons les différentes extensions de ce métamodèle, qui constituent les deux profils UML, nommés *Symphony Specifications* et *Symphony Analysis*. Nous abordons la métamodélisation des Objets Symphony en décrivant dans un premier temps des métamodèles indépendants du standard UML et ciblés chacun sur une facette particulière de notre problématique. L'intégration de ces concepts dans les profils UML idoines est réalisée dans un second temps.

7.2.1 Métamodèle de niveau spécification

Le premier métamodèle concerne les Objets Symphony, tels que construits lors de la phase de spécification organisationnelle et interactionnelle des besoins (c.f chapitre 5). La réalisation de cette phase entraîne la construction de trois modèles complémentaires : les modèles des Objets Métier, des Objets Interactionnels et des connexions entre Objets Métier et Objets Interactionnels. Bien que réalisés par différents acteurs du développement, ces modèles sont trois facettes du modèle global des Objets Symphony de niveau spécification, dont nous présentons le métamodèle conceptuel en figure 7.2.

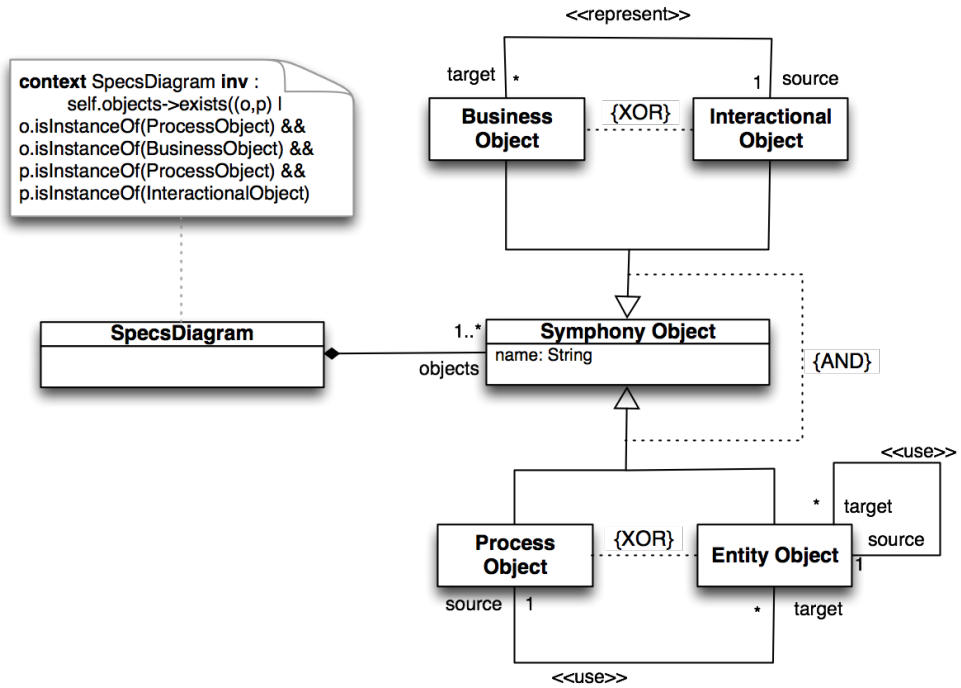


FIGURE 7.2 – Métamodèle conceptuel des Objets Symphony de niveau spécification

Ce métamodèle décrit les différentes distinctions orthogonales entre Objets Métier et Objets

Interactionnels d'une part, et Objets Entité et Objets Processus d'autre part. On y retrouve également les deux relations « utilise » et « représente », la première étant incluse dans le métamodèle d'UML, la seconde spécifique à Symphony étendue. Enfin, une contrainte OCL précise que, pour tout diagramme d'Objets Symphony, il est nécessaire de définir au moins un Objet Métier Processus et un Objet Interactionnel Processus.

Ces concepts sont intégrés au profil UML *Symphony Specifications*, détaillés en figure 7.3. Les Objets Symphony de niveau spécification correspondent à des paquetages vides, qualifiés selon les types orthogonaux Objet Métier/Objet Interactionnel et Objet Entité/Objet Processus. La relation « représente » est quant à elle définie comme une relation de dépendance, au même titre que la relation « utilise ». Une contrainte OCL limite son application : les éléments reliés doivent correspondre à des Objets Symphony Entité et les extrémités de la relation doivent être de sous-types (Objet Métier/Objet Interactionnel) différents.

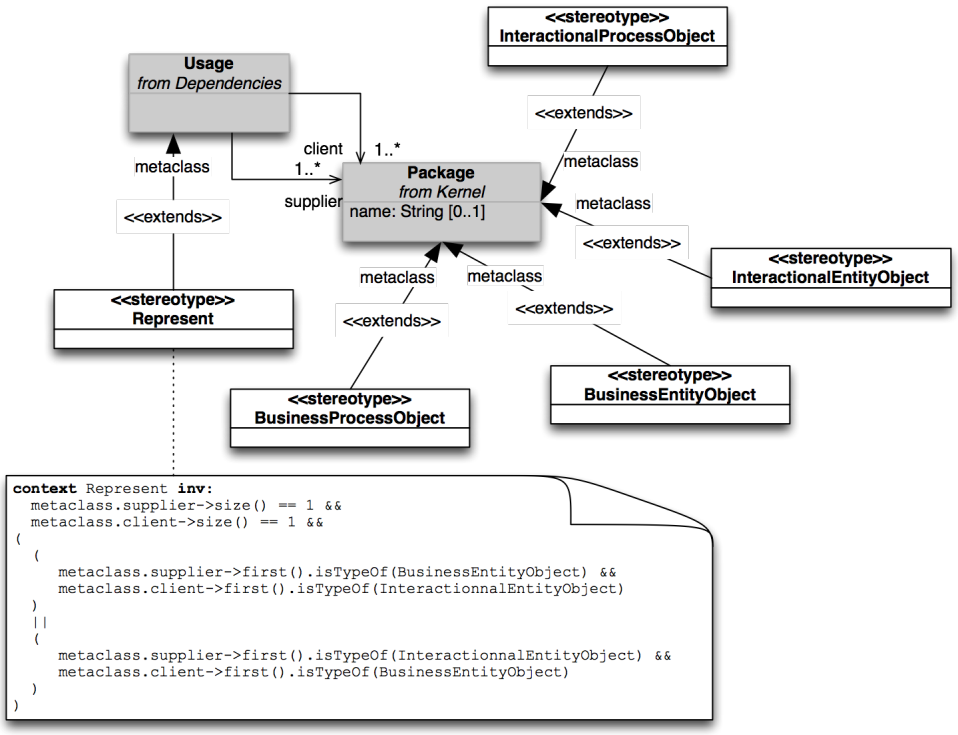


Figure 7.3 – Profil UML *Symphony Specifications*, centré sur la modélisation de niveau spécification

7.2.2 Métamodèle de niveau analyse

Les Objets Symphony sont raffinés lors de l'analyse en paquetages tripartites contenant des classes stéréotypées « Interface », « Maître », « Partie » et « Rôle », permettant d'identifier les responsabilités de chaque élément.

De même que précédemment, il est possible d'envisager la description du modèle d'analyse statique comme la composition de trois constructions partielles : le modèle statique des Objets Métier (traité précédemment par I. Hassine [57]), le modèle statique des Objets

Interactionnels et le modèle statique de Translation. Par souci de concision, nous nous concentrons ici sur le modèle global.

La figure 7.4 reprend le métamodèle global des Objets Symphony de niveau analyse. La métamodélisation des méthodes n'y est que sommairement abordée : assumons qu'elle est comparable à la structure décrite dans le métamodèle UML en figure 7.1. De même, par souci de lisibilité, les contraintes déjà exprimées dans le métamodèle de niveau spécification ne sont pas reproduites dans la figure 7.4. Le concept de classe Translation fait son apparition dans le métamodèle comme matérialisation de la relation représente. Autre particularité de ce métamodèle : à l'instar des relations « représente », l'utilisation des relations « utilise » est limitée aux classes « Rôle » et « Interface ».

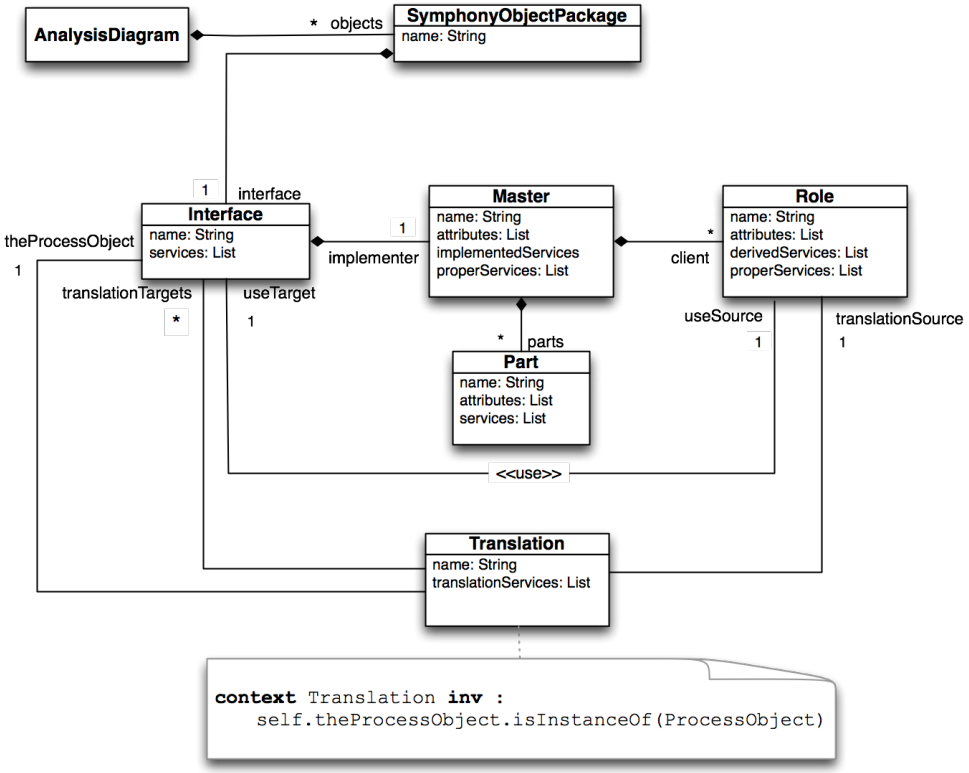


FIGURE 7.4 – Métamodèle simplifié des Objets Symphony de niveau analyse (à la suite de la description de la classe Translation)

Nous traduisons les concepts décrits de figure 7.4 dans les termes du profil UML *Symphony Analysis*, représenté en figure 7.5. L'analogie entre les deux représentations est triviale, toutefois un concept manque à cette métamodélisation : la notion d'annotation permettant d'identifier les méthodes des Objets Symphony susceptibles de déclencher des événements de Connexion Symphony en direction des classes de Translation (c.f chapitre 4, et que nous traitons ci-dessous.

7.2.3 Métamodèle d'annotation Symphony

La superstructure UML aménage la possibilité d'annoter tout élément d'un modèle UML, grâce au concept de commentaire (*Comment*, dans la figure 7.6). Il est ainsi envisageable

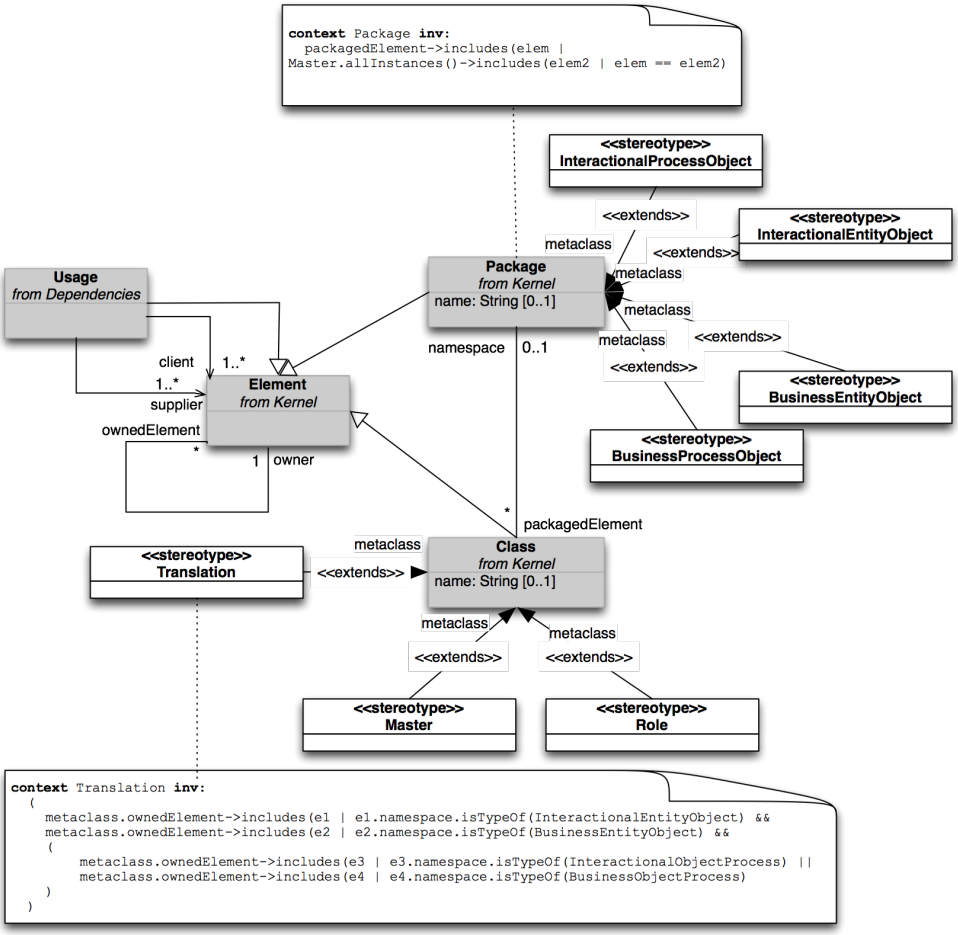


FIGURE 7.5 – Profil UML *Symphony Analysis*, centré sur la modélisation de niveau analyse

de commenter une méthode d'un diagramme de classes UML : une méthode est en effet considérée dans la superstructure UML comme une opération, concept étendant celui d'élément.

Nous avons étendu le concept de commentaire par celui d'annotation Symphony (*SymphonyAnnotation* sur le diagramme), tel qu'illustré en figure 7.6. Cette extension consiste simple-

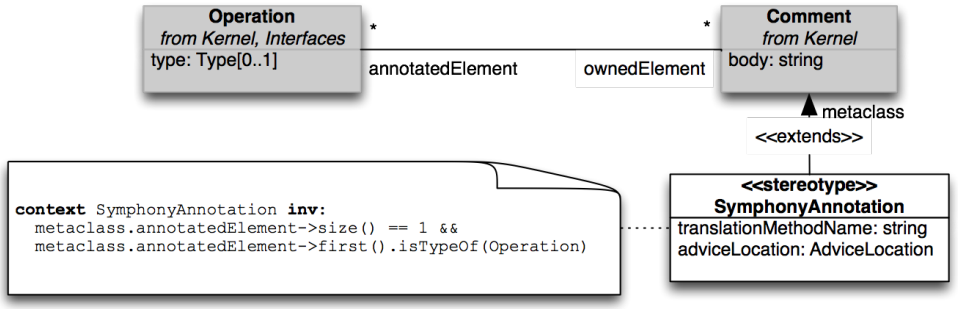


FIGURE 7.6 – Ajout au profil UML *Symphony Analysis*, intégrant la prise en compte des annotations

ment en la définition de deux attributs `translationMethodName` et `adviceLocation`, la valeur du premier correspondant au nom de la méthode de la Translation invoquée par Sonata lorsque la méthode ciblée par l'annotation est elle-même appelée, le second identifiant la position du greffon. Une contrainte OCL réduit la portée de l'annotation Symphony en précisant que celle-ci ne peut être appliquée qu'à une opération UML (c'est-à-dire, une méthode de classe). Ces deux éléments sont ajoutés au profil UML *Symphony Analysis* dédié aux modèles Symphony de niveau analyse.

Les profils UML ainsi construits nous permettent de définir un ensemble de transformations pour la vérification des contraintes de cohérence, et la génération partielle de modèles d'analyse et de conception.

7.3 Transformations et cohérence des Objets Symphony de la spécification à conception

La figure 7.7 reprend les différentes transformations appliquées, au cours du cycle de développement, sur les Objets Symphony. Les produits les plus à droite de la figure signalent les artefacts entrés par l'utilisateur. Les produits de la colonne centrale de la figure indiquent les artefacts produits par les transformations.

Bien que les modèles impliqués soient nombreux, il convient de noter que nous distinguons les différents stades de complétion d'un produit en autant d'artefacts de conception. Nous décomposons également les différentes étapes de transformation, dont plusieurs sont réalisées en une seule opération à l'exécution. Sur la figure 7.7, ce cas concerne les enchaînements de transformations basées sur des artefacts eux-mêmes produits par transformation. Par exemple, les opérations « Vérification de la cohérence du modèle global » et « Génération du modèle d'analyse des Objets Symphony » sont exécutées ensembles.

Dans les paragraphes qui suivent, nous traitons d'abord les transformations de raffinement et les vérifications de cohérence appliquées entre les niveaux de spécifications et d'analyse, puis nous détaillons les transformations construites entre les niveaux d'analyse et de conception. Les transformations vers le modèle d'implémentation, c'est-à-dire le code d'implémentation, relèvent de la problématique de génération de code Java à partir de modèles UML largement abordés dans la littérature, laquelle n'est pas traitée dans nos travaux.

Enfin, ajoutons que l'implémentation des règles de transformation est décrite en annexe B.

7.3.1 Transformation des modèles de niveau spécification

Nous abordons ici la génération du modèle d'analyse des Objets Symphony, à partir des modèles de niveau spécification, ainsi que les vérifications de cohérence précédent et suivant cette transformation.

7.3.1.1 Génération du modèle global des Objets Symphony

La génération du modèle global des Objets Symphony consiste en une fusion des trois modèles partiels de niveau spécification (modèle des Objets Métier, modèle des Objets Interactionnels et modèle des Connexions Symphony). Cette transformation se base sur une

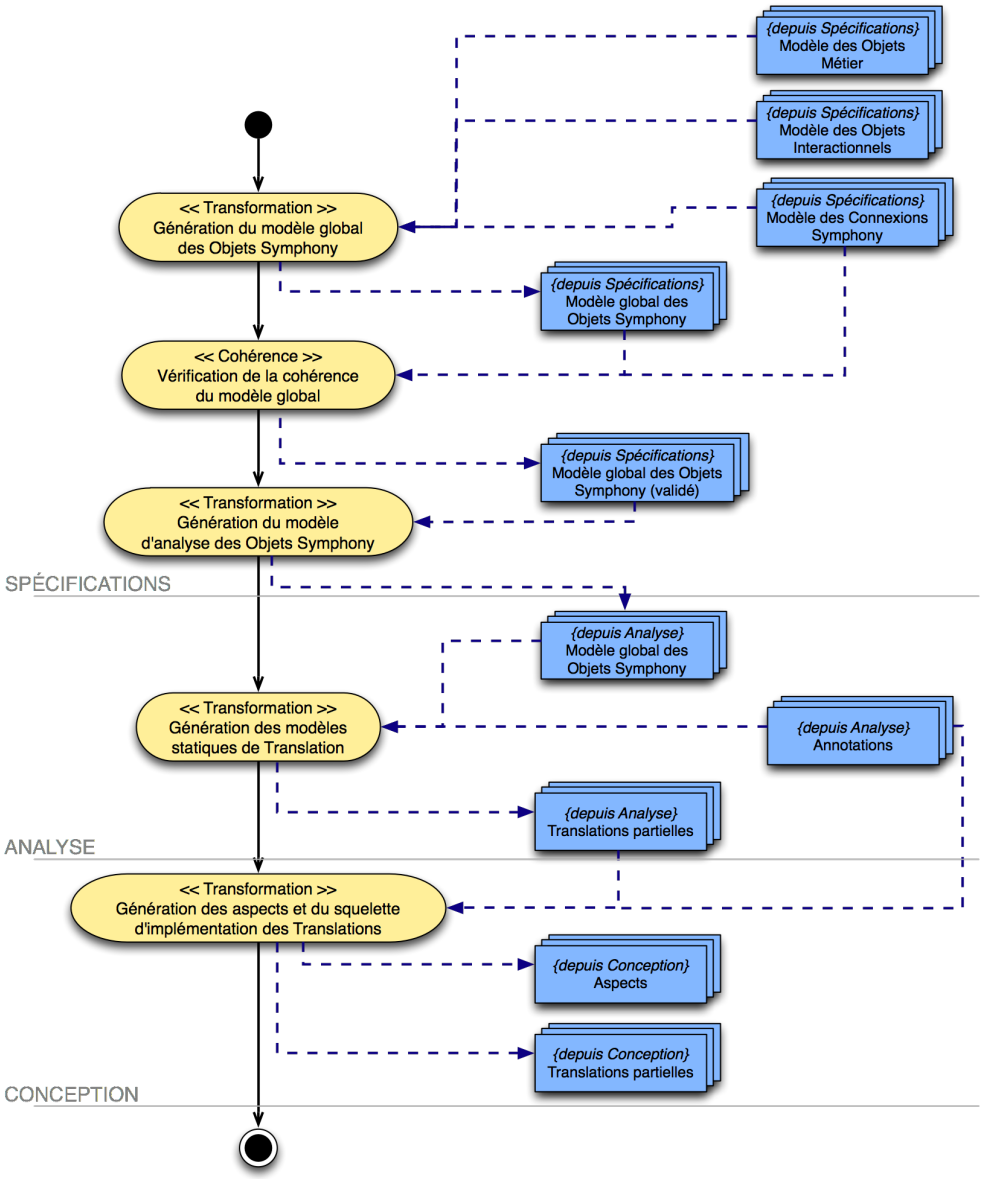


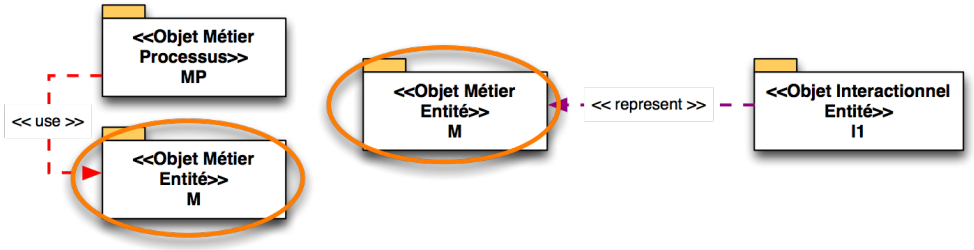
Figure 7.7 – Étapes de transformation et raffinement des Objets Symphony au cours du cycle de développement

recherche de correspondances lexicales entre les noms des Objets Symphony décrits dans chaque modèle.

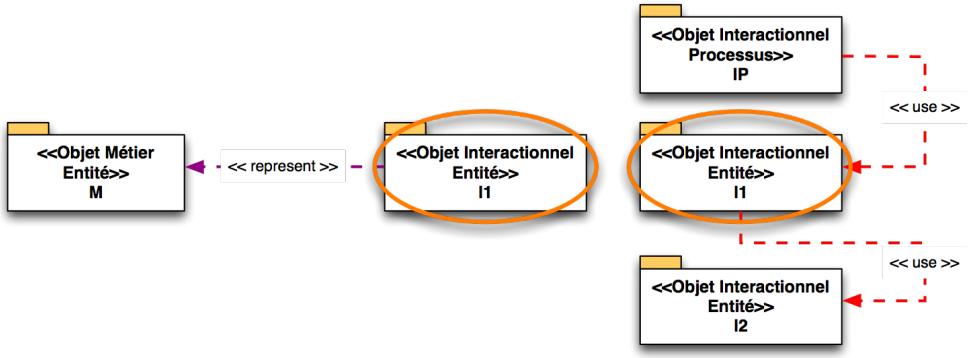
7.3.1.2 Vérification de la cohérence du modèle global

La vérification de cohérence sur les modèles partiels de niveau spécification suit la transformation unifiant les trois modèles partiels, et précède la génération partielle du modèle d'analyse. Il s'agit de vérifier la propriété d'injection entre le modèle des Connexions Symphony et les modèles des Objets Métier et Interactionnels, soit : pour chaque Objet Symphony du modèle des Connexions Symphony, il doit exister un Objet Symphony de même nom

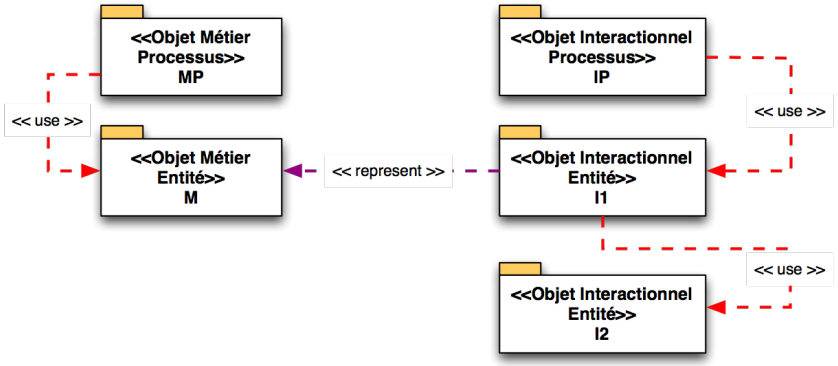
dans le modèle des Objets Métier ou le modèle des Objets Interactionnels. La figure 7.8 illustre ce cas de figure : étant donné l'Objet Métier Entité « M » du modèle des connexions Symphony, celui-ci doit être présent dans le modèle des Objets Métier. De même, étant donné l'Objet Interactionnel Entité « I1 », celui-ci doit apparaître dans le modèle des Objets Interactionnels.



(a) Cohérence entre le modèle des Objets Métier et le modèle des connexions Symphony



(b) Cohérence entre le modèle des connexions Symphony et le modèle des Objets Interactionnels



(c) Modèle global des Objets Symphony, après fusion des modèles partiels

FIGURE 7.8 – Exemples de cohérences lexicales entre modèles partiels et modèle global de niveau spécification

7.3.1.3 Génération du modèle d'analyse des Objets Symphony

Les correspondances entre éléments des métamodèles de spécification et d'analyse sont évidentes et décrites dans le tableau 7.1. On pourra se référer, à titre d'illustration, au raffi-

nement de l'Objet Métier Entité « Dommage », décrit au chapitre 3, page 93, ou à celui de l'Objet Interactionnel Entité « Marqueur », décrit au chapitre 4, page 113. Les conventions de nommage de niveau analyse ayant déjà été abordées au chapitre précédent, nous les éludons dans les descriptions des transformations ci-dessous.

Tableau 7.1 – Règles de transformation des Objets Symphony, de la spécification à l'analyse

Éléments source	Éléments générés [a]
• Paquetage UML	• 1 classe « Interface » • 1 classe « Maître »
• Relation « Utilise »	• 1 classe « Rôle » pour l'objet client, liée à la classe « Maître », • 1 relation « Utilise » entre la classe « Rôle » du client et la classe « Interface » du fournisseur
• Relation « Représente »	• 1 classe « Translation » disposant des références des objets source et cibles. • La source de la relation « Représente » est déplacée sur la classe « Rôle » de l'objet utilisant l'objet source, la cible est déplacée sur l'Objet Processus détenant l'objet cible

a. pour chaque occurrence d'un élément source

Une fois les méthodes des domaines métier et interaction définies, le concepteur ajoute les annotations définissant le déclenchement des événements de Connexion Symphony. Grâce aux indications présentées dans les annotations, il est alors possible de générer le squelette des classes de Translation.

7.3.2 Transformations des modèles de niveau analyse

Au-delà des modèles d'analyse, le processus de raffinement des modèles Symphony intègre les composants techniques et les contraintes technologiques analysées dans la branche technique du cycle de développement. En écho au chapitre précédent, nous traitons dans cet intitulé les transformations permettant d'automatiser la conception et l'implémentation des composants Sonata. On trouvera page 166 un exemple de raffinement d'Objet Symphony, du niveau analyse au niveau conception. Nous nous focalisons sur les transformations traitant des particularités des modèles Symphony de niveau conception et des composants Sonata, résumées ci-dessous :

- les classes « Maître » implémentent les interfaces de marquage `EntityObject` ou `ProcessObject`, les classes « Rôle » implémentent quant à elles l'interface de marquage `SymphonyRole` (c.f. chapitre précédent) ;
- par défaut, les classes « Rôle » implémentent les méthodes de l'objet fournisseur, selon la patron « Délégué » [52], c'est-à-dire que chaque méthode utilise la référence à l'objet fournisseur, créée lors du tissage du service `SymphonyRoleServices` pour transmettre l'appel de méthode à l'Objet Symphony fournisseur ;
- l'intergiciel Sonata est configuré pour la mise en œuvre des Connexions Symphony ; un fichier de configuration recensant les Objets Symphony impliqués dans les connexions, ainsi que les classes de Translation, est généré ; les aspects tissant les Connexions Symphony

dans les Objets Symphony idoines sont également générés.

7.3.2.1 Génération des aspects

Nous avons présenté au chapitre précédent le mécanisme sur lequel s'appuie l'intergiciel Sonata pour transmettre un appel de méthode déclencheur d'un événement de connexion vers la Translation idoine. Ce mécanisme utilise des aspects, dont nous avons évoqué à la fois la structure et les règles de construction systématiques. Ceux-ci se prêtent donc tout particulièrement à un effort d'automatisation, ciblé sur la transformation des modèles d'analyse (modèle statique d'Objets Symphony et annotations) vers les modèles de conception et le code d'implémentation. Contrairement aux transformations précédentes, la génération d'aspects proposée ici est complète.

La génération des aspects, nécessaires à l'intergiciel Sonata, s'appuie sur les annotations apposées aux modèles issus de l'analyse statique et sur un métamodèle très simplifié des aspects tels qu'implémentés dans le projet AspectJ [ASP].

Concrètement, tout aspect destiné à la connexion entre les Objets Symphony et les classes de Translation, par l'intermédiaire de l'Invoker (c.f. chapitre précédent) est structuré selon le squelette suivant :

```
1  package control.communication;
2  <liste_importations>
3
4  public aspect <nom_aspect> {
5
6    // Définition du point de jonction
7    pointcut <nom_point_de_jonction>(): execution(public * <
         nom_classe_role>.*(..)) ;
8
9    // Définition du greffon
10   <localisation_aspect>: execution(<signature_methode_source>)
         <selection_arguments> {
11     try{
12       // Construction de l'événement de connexion
13       Invoker.instance.createRequest(<instance_objet_appele>, <
             nom_methode_cible>);
14       <chargement_arguments>
15
16       // Déclenchement de l'événement
17       Invoker.instance.sendRequest();
18     } catch (Exception e) {/* Gestion des exceptions */}
19   }
20 // Autres connexions
21 }
```

Les concepts saillants de ce squelette d'aspects sont exprimés dans le métamodèle décrit en figure 7.9. Ce dernier recense en particulier les points de variation (entre crochets dans le squelette ci-dessus) à partir desquels il est possible de générer un aspect complet. Nous les reprenons ci-dessous :

- la définition de points de jonction (permettant de restreindre le champ d'application de l'aspect aux seuls appels effectués sur la classe ciblée par l'aspect) ;

- l'identification de l'objet concernée par l'appel de méthode sur laquelle est tissée l'aspect ;
- l'identification des arguments utilisés comme paramètres de la méthode sur laquelle est tissée l'aspect ;
- l'identification de la méthode sur laquelle tisser l'aspect ;
- la sélection du déclenchement de l'événement de connexion avant ou suivant la résolution de la méthode sur laquelle s'effectue le tissage.

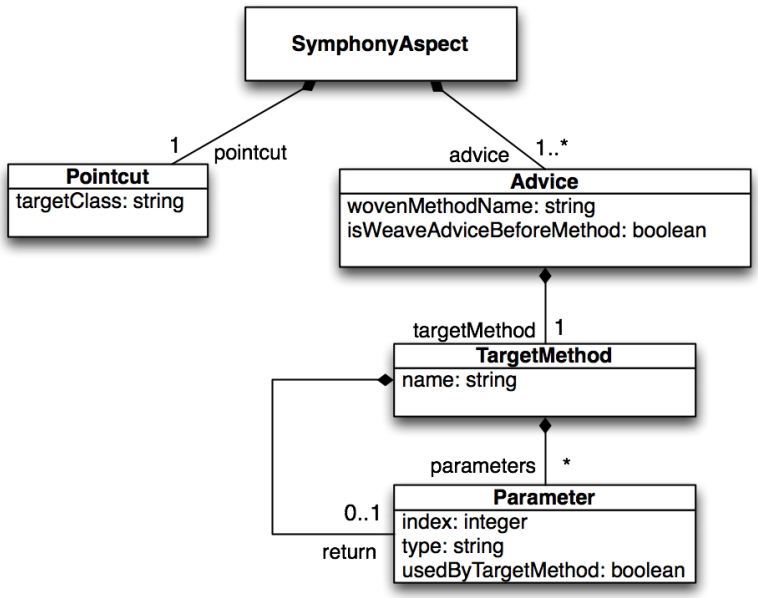


Figure 7.9 – Métamodèle des aspects AspectJ simplifié

Les éléments tirés du modèle d'analyse annoté sont :

- La signature de la méthode annotée,
- Le nom de la méthode censée être appelée par la Translation (décrite dans l'annotation), ainsi que la localisation du greffon sur la méthode annotée (suivant ou précédent l'appel à la méthode),
- Les noms des différentes classes du modèle d'analyse (utilisé pour la gestion des importations, dans l'aspect généré)

Un exemple d'aspect généré est décrit au chapitre précédent, page 168.

L'utilisation conjointe des annotations et de la modélisation simplifiée du formalisme utilisé par AspectJ nous permet ainsi de définir un mécanisme de transformation simple dont le résultat est un fichier décrivant l'aspect, confirme du point de vue du compilateur AspectJ. Un fichier aspect est généré par classe Translation, au sein duquel sont regroupés les greffons permettant d'effectuer l'appel aux méthodes de la Translation, ainsi que le point de jonction limitant le tissage de l'aspect à une seule classe.

Complément indispensable du processus de transformation de modèles, la section suivante présente une approche de traçabilité centrée sur les Objets Symphony.

7.4 Traçabilité des transformations

Nous proposons une forme simple de traçabilité, basée sur les modèles d'Objets Symphony construits au cours des phases de spécification, d'analyse et de conception. Bien que d'un périmètre limité, cette approche de traçabilité a l'avantage notable de ne demander aucune intervention de la part du concepteur (c'est-à-dire, aucune identification ou marquage particuliers des artefacts). Asuncion et al. [3] ont en effet noté que les approches de traçabilité requérant de nombreuses manipulations manuelles sont un frein de productivité que leurs bénéfices compensent rarement.

Notre approche de traçabilité consiste, dans le contexte de la modélisation des Objets Symphony, à proposer plusieurs vérifications de cohérence entre :

- les modèles partiels exprimés au niveau spécification et le modèle global des Objets Symphony ;
- le modèle global des Objets Symphony et le modèle d'analyse ;
- le modèle d'analyse et le modèle de conception.

La traçabilité ainsi obtenue, résumée en figure 7.10, permet de maintenir la cohérence entre les différents modèles d'Objets Symphony, malgré les modifications et raffinements successifs opérés par les concepteurs, et de remonter chaque produit décrit aux niveaux conception ou analyse à un Objet Symphony de niveau spécification. Bien que limitée à un nombre réduit d'artefacts du développement, cette mise en œuvre de traçabilité permet néanmoins de réduire le spectre des produits non tracés aux phases de spécification et d'étude préalable, et facilite donc le suivi des choix de conception et des raffinements et modifications de produits..

Les différentes opérations de vérification de cohérence constituant le processus de traçabilité décrit en figure 7.10 sont détaillées en annexe B.

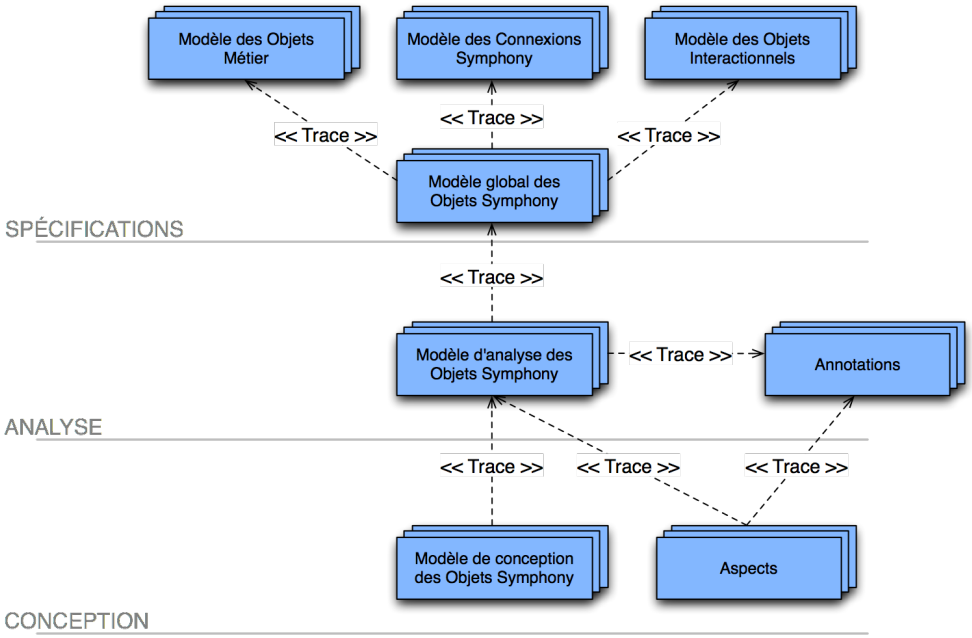


FIGURE 7.10 – Opérationnalisation de la traçabilité des Objets Symphony

Les technologies utilisées pour la description des transformations et vérifications de cohérence sont décrites dans la section suivante.

7.5 Outils pour la transformation de modèles

Les transformations décrites dans ce chapitre sont implémentées à l'aide du langage ATL (*ATLAS Transformation Language*) et exécutée grâce à la machine virtuelle ATL. Le langage ATL est une implémentation réalisée au sein des groupes de recherche LINA et ATLAS (INRIA Rennes) [65] du standard pour la transformation de modèles QVT (*Query/View/Transformation*) proposé par l'OMG (*Object Management Group*), organisme responsable entre autres des standards UML et Unified Process.

ATL est un langage fonctionnel, dont le concept central est le modèle et dont le métamodèle est également conforme au MOF. Les transformations ATL sont exprimées d'un ou plusieurs modèles sources vers un modèle cible, ou bien d'un ou plusieurs modèles sources vers un résultat textuel tel que du code d'implémentation ou la réponse à une requête (par exemple, une requête évaluant la cohérence d'un modèle par rapport à un ensemble de contraintes de construction).

La figure 7.11 résume les différents concepts de la transformation de modèles, appliquée aux technologies Eclipse/ATL. Les métamodèles source et cible sont définis tous deux à l'aide du formalisme Ecore [Eco]. Le formalisme Kermeta [Ker], dont l'avantage est d'être indépendant de toute technologie d'implémentation, est également utilisable pour la définition de métamodèles.

Que ces métamodèles soient définis à l'aide des langages Ecore ou Kermeta, ils restent conformes au méta-métamodèle MOF (*Meta-Object Facility*), qui définit le paradigme de modélisation Entité–Relation. Les instances des métamodèles sont exprimées au format XMI (*XML Metadata Interchange*) [88], un dialecte XML généralement utilisé pour l'échange de modèles UML.

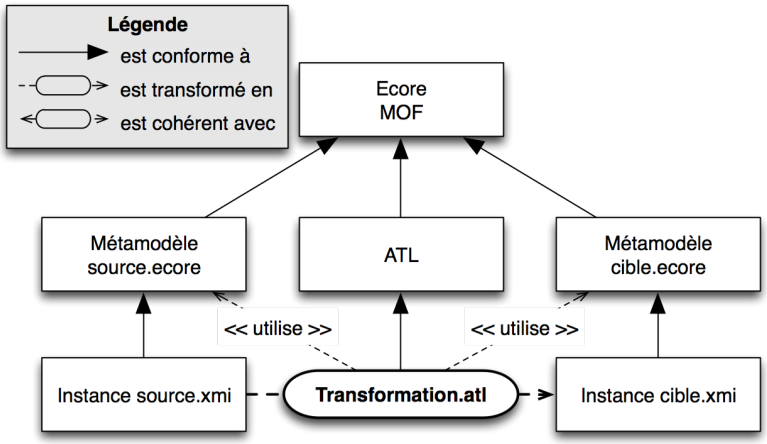
À ce jour, les métamodèles des Objets Symphony (niveaux spécification et analyse) et des aspects ont été implémentés. Plusieurs itérations ont été réalisées sur les règles de transformation et de cohérence, au fil de l'évolution des métamodèles. Sur les cinq règles de transformation et cohérence relatives à la génération des Objets Symphony, deux sont à des stades avancés de finalisation ; sur les huit règles de cohérence relatives à la traçabilité, la moitié est actuellement implémentée.

Par ailleurs, la définition des deux profils UML Symphony est réalisée à l'aide du *plugin* Topcased [Top] de l'environnement intégré de développement Eclipse [Ecl]. Leur implémentation est également en cours de finalisation.

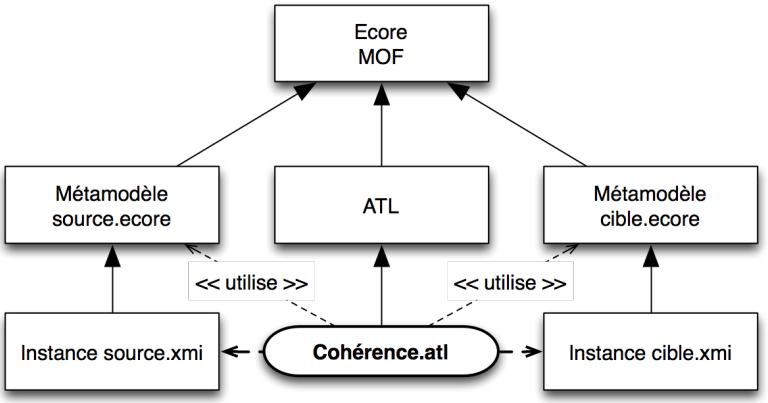
7.6 Synthèse

Outre la documentation du processus de développement, décrite au chapitre 4, et l'instrumentation à l'exécution, décrite au chapitre 6, certains modèles de la méthode Symphony font également l'objet d'une opérationnalisation.

Contribution originale de la méthode, les Objets Symphony sont métamodélisés et intégrés



(a) Transformation de modèle



(b) Vérification de cohérence entre modèles

FIGURE 7.11 – Modèles génériques de manipulation de modèles

au langage UML grâce à l'usage de profils. Ces derniers concernent la modélisation des Objets Symphony aux niveaux spécification et analyse.

Ces métamodèles sont engagés d'une part dans l'automatisation des activités systématiques de la conception (fusion de modèles partiels et raffinement de modèles), et d'autre part dans l'organisation de la traçabilité des raffinements des Objets Symphony. Bien que partielle, cette approche de traçabilité permet de réduire le spectre des produits non tracés et facilite donc le suivi des choix de conception et des raffinements et modifications de produits.

*

* *

Le prochain chapitre, qui est aussi le dernier de ce manuscrit, expose les évaluations de deux aspects majeurs de notre méthode : le code d'implémentation produit suivant l'application de la méthode, et le processus de conception.

CHAPITRE 8

Validation

> Sur place, tu vois naître les désordres, auxquels tu peux parer aussitôt ; mais à distance, tu en es informé lorsqu'ils sont si grands qu'il n'y a plus remède.
>
> *Le Prince*
> NICOLAS MACHIAVEL

BIEN QUE la méthode Symphony originelle ait été appliquée avec succès à de nombreux projets de développement logiciel, l'évaluation de notre extension de la méthode est essentielle. Cette évaluation implique une nouvelle fois la prise en compte des quatre composants fondamentaux : le processus, les langages, les modèles et les outils ; à ceux-ci nous rajoutons une nouvelle dimension : l'acceptation par les utilisateurs.

Ce dernier chapitre expose les enjeux de l'évaluation des méthodes de développement, et présente deux expériences ayant pour objectif de valider les composants de la méthode et l'acceptation par les utilisateurs.

8.1 Évaluation des méthodes de développement

La « correction » d'une méthode n'est en effet pas suffisante pour justifier son utilisation : plusieurs paramètres psychologiques et sociologiques guident les réactions – positives ou négatives – de potentiels utilisateurs mis en présence de nouvelles technologies, ainsi que le démontre le modèle d'acceptation technologique (*Technology Acceptance Model*) de Davis [40].

Cependant, peu de travaux abordent la validation des méthodes de développement. En effet, là où les modèles disposent de plusieurs cadres théoriques ou pratiques d'évaluation (par exemple, le standard ISO 9000, J. Krogstie [71], ou Lange & Chaudron [73], pour n'en citer que quelques uns), les méthodes de développement sont généralement évaluées à l'aide du seul modèle d'acceptation technologique de Davis [40] (le lecteur pourra se référer, à titre

d'exemple, aux travaux de Chan et Thong [23] sur l'acceptation des méthodologies agiles), ou bien à l'aune de leur adoption dans l'industrie.

Néanmoins, dans la perspective d'améliorer l'évaluation de la « réussite » des méthodologies, D. L. Moody [79] propose d'intégrer au modèle d'acceptation technologique celui du pragmatisme méthodologique. Les méthodes de développement sont alors évaluées selon deux axes orthogonaux :

1. Le pragmatisme méthodologique prône l'évaluation de l'effectivité (*effectiveness*), mesurant la qualité du produit résultant de l'application de la méthode, et l'évaluation de l'efficience (*efficiency*), mesurant la complexité de la tâche requise pour réaliser le produit,
2. Le modèle d'acceptation technologique considère :
   - la perception de la facilité d'utilisation (*perceived ease of use*), c'est-à-dire *« le degré auquel la personne croit que l'utilisation du système sera dénuée d'effort »* [40],
   - la perception de l'utilité (*perceived usefulness*), c'est-à-dire *« la probabilité subjective de la personne selon laquelle l'utilisation du système améliorera ses performances »* [40],
   - l'intention d'utiliser (*intent to use*), c'est-à-dire *« la mesure selon laquelle la personne compte utiliser le système »* [40].

La combinaison de ces deux modèles par Moody dans le cadre théorique du « modèle d'évaluation des méthodes » (*Method Evaluation Model*) est synthétisée en figure 8.1. Celle-ci fait émerger les dimensions objective (ou réelle) et subjective (c'est-à-dire, perçue) de l'adoption d'une méthode, ainsi que les liens entre celles-ci, confirmés expérimentalement. Il y est notamment mis en évidence le rôle central de la perception de la facilité d'utilisation dans le processus d'acceptation technologique.

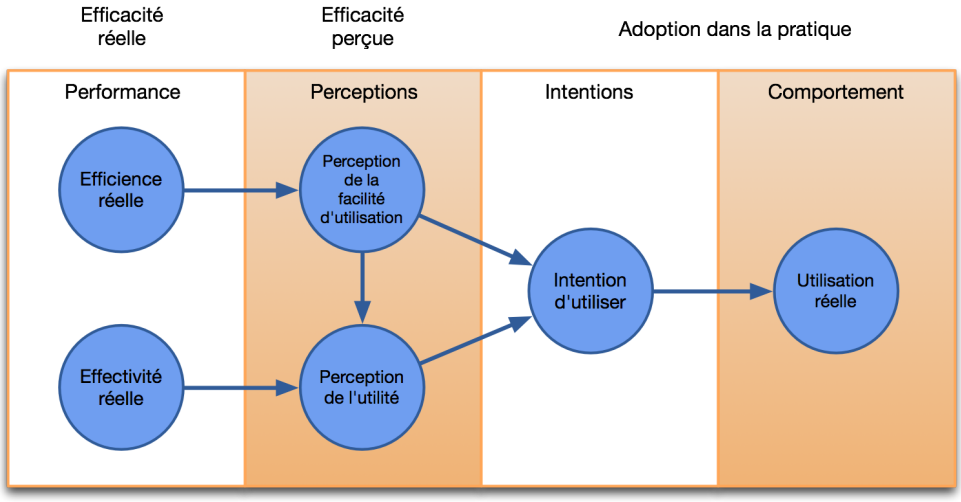


FIGURE 8.1 – Modèle d'évaluation des méthodes, d'après Moody [79]

Le recours à ce cadre théorique de validation nous a permis d'identifier les axes selon lesquels nous devons évaluer notre méthode. Dans le cadre de cette thèse, nous avons ainsi analysé :

1. La perception de la facilité d'utilisation de la méthode, en impliquant des utilisateurs dans un fragment du cycle de développement (phases de spécification et d'analyse). L'évaluation devait concerner d'abord le processus, les langages et les modèles, mais les résultats ont également abordé une partie de l'outillage.
2. L'effectivité réelle de la méthode, en considérant différentes implémentations réalisées suivant les recommandations de la méthode. L'évaluation s'est concentrée sur les modèles de niveau analyse et conception.

Ces estimations se sont inscrites dans un processus de définition itérative des composants de notre méthodologie. Les résultats présentés dans ce chapitre ne reflètent donc pas l'état le plus récent de nos contributions, mais esquissent en creux l'orientation et les objectifs de la dernière itération que nous ayions réalisée. Au jour où nous remettons ce manuscrit, il semble évident que la méthode Symphony étendue mériterait encore évolutions et évaluations.

Nous détaillons dans la suite de ce chapitre deux expérimentations, dont nous présentons, pour chacune, le protocole, le déroulement des évaluations, les résultats obtenus et enfin une discussion sur les aspects de la méthode évalués par l'expérimentation.

8.2 Évaluation de l'efficience de la méthode : collaboration entre utilisateurs experts

Les phases en amont de la méthode Symphony intègrent un nombre important de collaborations entre spécialistes GL et IHM. La validation de ces activités, dont le principe est la confrontation de pratiques relativement différentes, est cruciale pour l'estimation de la qualité globale de la méthode. Nous avons donc entrepris d'évaluer les phases de spécification conceptuelle et de spécification organisationnelle et interactionnelle des besoins. Spécifiquement, nous avons estimé, dans le contexte favorable d'une équipe de développement constituée d'experts des domaines du GL et de l'IHM, si la méthode Symphony étendue permet aux acteurs GL et IHM du développement de construire les spécifications organisationnelle et interactionnelle des besoins de manière simultanée et indépendante. La méthode doit également permettre de confronter les résultats de ces activités et d'établir une cohérence entre ceux-ci.

8.2.1 Protocole expérimental

L'expérience implique quatre groupes de deux personnes, l'une étant expert IHM, l'autre expert GL. Tous sont des universitaires habitués au travail en équipes de même compétence, mais peu habitués à confronter des conceptions logicielles issues de pratiques différentes.

Les différentes phases de l'expérience s'étalent sur une semaine. Trois séances de travail ont été organisées. La première séance est axée sur une présentation rapide aux huit sujets de la méthode Symphony étendue, du cas d'étude et de son cahier des charges. Ce dernier consiste en une application d'aménagement collaboratif d'espaces pour l'organisation d'événements de type salons, conventions, congrès etc., pour laquelle nous attendons une solution interactive post-WIMP, de type table augmentée notamment.

Un intervalle de quatre jours a séparé la seconde séance de la première, pendant lequel les participants ont travaillé seuls. Durant la deuxième séance, il leur a été proposé de faire la mise en correspondance de leurs modèles.

Afin de les assister pendant ces quatre jours, différentes ressources ont été mises à la disposition des participants : articles, experts de la méthode (dont l'auteur), et l'aide en ligne AGAP Lite (c.f chapitre 4), alors en cours de développement.

De plus, un entretien a été mené avec les participants à l'issue de la deuxième séance. Sept points de la méthode ont été évalués, répartis selon les trois composants des méthodes : les processus, les langages et modèles, et l'outillage.

Processus Le processus de spécification a été évalué, ainsi que l'enchaînement des activités,

Langages et modèles L'utilisation des formalismes CTT et ASUR a été évaluée, ainsi que la représentation des concepts métier sous la forme d'Objets Métier et la représentation des concepts de l'interaction sous la forme d'Objets Interactionnels

Outillage L'outil d'aide en ligne AGAP Lite a été évalué, ainsi que l'utilisation du langage UML pour la spécification des patrons (notamment les diagrammes d'activités)

La troisième séance a consisté en une présentation des spécifications élaborées par chaque binôme, suivi d'un entretien final sur les points positifs et les points négatifs de la méthode et de l'aide en ligne, selon la même répartition qu'à la deuxième séance. Il a également été proposé aux participants de suggérer des améliorations.

8.2.2 Résultats

Parmi les huit sujets restants, un n'a pas participé à l'entretien final. Certains participants n'ont pas réalisé l'ensemble des produits attendus au cours de l'expérience. Sur les quatre groupes, deux n'ont pas réalisé de mise en relation entre Objets Métier et Objets Interactionnels (un groupe n'a pas décrit d'Objet Symphony, un autre groupe n'a décrit que des Objets Interactionnels). Néanmoins, tous les participants ont tenté de mettre en cohérence certains produits issus des domaines métier et interaction. Trois groupes ont vérifié la cohérence entre PM, PMC et modèles de tâches utilisateurs ; les deux groupes ayant réalisé des OMs et OIs ont effectué une mise en relation ; enfin, les scenarii ont également été un repère quant à la cohérence du système pour deux groupes. Enfin, trois groupes sur quatre ont décrit une interaction post-WIMP.

Les résultats décrits ci-dessous sont issus des entretiens qualitatifs menés avec les sept participants restants.

L'ensemble des sujets a jugé la méthode Symphony intéressante et satisfaisante quant aux échanges qu'elle facilite entre spécialistes du Génie Logiciel et de l'IHM.

Bien que la majorité des participants a estimé que la méthode peut faire gagner du temps (6/7 sujets), certains ont au contraire pensé que celle-ci *« peut être longue »* (3/7 sujets). Les raisons invoquées quant au gain de temps sont :

- les concepteurs s'engagent dans le développement à partir d'une même vision commune ;
- la méthode contraint les concepteurs à se ramener à un niveau d'abstraction *« adéquat »*, mais qui, selon l'un des spécialistes GL, peut apparaître laborieuse lors des activités de collaboration.

De même, l'ensemble des participants estime que Symphony peut éviter des erreurs, car elle permet :

- une meilleure compréhension entre personnes de domaines différents ;
- un guidage dans les différentes étapes, détaillant les produits à réaliser et leur contenu ;
- une approche systématique permettant de soulever des questions *« non habituelles »*.

Toutefois, l'un des sujets relève le manque de *« phases de résolution de conflit »*.

Une majorité des sujets serait prête à décrire le fonctionnement de Symphony à un tiers et à l'adopter pour ses propres développements (6/7 sujets). De même, l'ensemble des sujets conseillerait l'utilisation de la méthode. Les motivations citées sont :

- une *« méthode itérative pilotée par scénarios ou cas d'utilisation »* ;
- un cadre précis pour la collaboration entre les différents spécialistes ;
- une séparation des préoccupations GL et IHM.

Les participants pensent que la méthode devrait être déployée sur des projets de taille importante (impliquant une dizaine de personnes).

Du point de vue de l'efficience de la méthode, un des sujets la conseillerait car il la juge *« simple, documentée et rapide à prendre en main »*. Cependant, de manière générale, les sujets considèrent qu'une phase d'apprentissage est nécessaire, celle-ci étant comprise entre quelques heures et plusieurs jours.

Au cours de l'exercice, les sujets ont souhaité obtenir des conseils. Ils ont pour cela mobilisé des ressources extérieures : Internet (5/7 sujets), des collègues de travail (2/7) et une thèse (1/7). Ils ont utilisé l'aide en ligne fournie par AGAP Lite de manière régulière (6/7). La clarté de cette aide est jugée moyenne (note moyenne de 6,9/10), plus largement dû à un problème de navigabilité (note moyenne de 5,6/10) qu'à un problème de vocabulaire (noté 7,1/10). L'un des sujets résume le problème de navigabilité comme suit : *« la navigation mérite vraiment un schéma synthétique de [la hiérarchie] des patrons et des flux de documents »*.

L'ensemble des remarques formulées au cours de la deuxième séance et lors de l'entretien final est résumé en annexe C. À l'issue de la dernière séance, les sujets ont indiqué deux améliorations à apporter à la méthode et à l'aide en ligne.

Concernant la méthode, après factorisation les huit points soulevés sont :

1. Expliciter les points où il est nécessaire de mettre en cohérence les diagrammes GL et IHM,
2. Adapter la méthode à la taille des projets,
3. Intégrer un glossaire des termes de la méthode ; dans l'ensemble, favoriser un langage clair et concis,
4. Décrire (pour les spécialistes IHM) la correspondance entre processus métier et modèle de tâches utilisateur,
5. Améliorer la phase de choix du type d'interaction,
6. Décrire en préambule les prérogatives de chaque spécialiste,
7. Expliciter la hiérarchie des Objets Métier et Interactionnels (par exemple à l'aide de métamodèles),
8. Proposer des modèles de présentation pour chaque produit de la méthode.

Concernant l'aide en ligne, après factorisation les améliorations suggérées sont :

1. Proposer une visualisation globale de la méthode,
2. Proposer une visualisation globale du cycle de vie des produits au cours du cycle de développement,
3. Proposer un repérage de l'activité en cours par rapport à l'ensemble de la méthode,
4. Fournir un plus grand nombre d'exemples, voire fournir un dossier complet en exemple,
5. Réduire la quantité de texte et privilégier les schémas,
6. Amélioration la présentation de l'outil ainsi que le système de navigation.

8.2.3 Discussion

Dans leur ensemble, les résultats de l'expérience tendent à démontrer que la méthode a effectivement permis aux spécialistes du GL et de l'IHM de collaborer pour confronter leurs visions des spécifications.

Certaines remarques soulignent la pertinence des principes fondamentaux décrits au chapitre 4. Ainsi, la documentation précise des activités de collaborations entre spécialistes du métier et de l'interaction apparaît comme essentielle. L'utilisation des scenarii et des Objets Symphony comme modèles pivots entre ceux deux pratiques semble également pertinent. De plus, le manque d'outils – notamment destinés à modéliser les diagrammes ASUR et CTT, pour lesquels il n'existe que deux outils accessibles sur internet, parfois difficilement – et les défauts des outils existants – l'aide en ligne, tout particulièrement – ont été fortement ressentis, ce qui confirme leur rôle central dans le cycle de développement.

D'autres remarques suggèrent d'adapter la méthode à la taille du projet, en omettant certaines étapes d'analyse du métier ou des utilisateurs lorsque le projet est une application simple destinée à un seul utilisateur, par exemple. Outre les étapes explicitement notées comme optionnelles, l'ajustement du cycle de développement aux caractéristiques du projet est une responsabilité du chef de projet, rôle absent de l'évaluation. Dans un contexte productique, celui-ci aurait toute latitude de modifier l'organisation initiale des activités, à condition de préserver la cohérence et la traçabilité des produits.

Plusieurs participants, issus du domaine de l'IHM, proposent de décrire un modèle de tâches utilisateur comme traduction des diagrammes d'activités UML décrivant le processus propre à chaque activité. Cette approche nous semble contradictoire avec l'objectif de ces diagrammes, pour les raisons suivantes :

- Les modèles de tâches, en particulier CTT, ne permettent pas de décrire explicitement les produits résultants d'une activité ou requis par une activité, contrairement aux diagrammes d'activités UML,
- Les modèles de tâches sont généralement centrés sur un seul utilisateur ; au contraire, il est nécessaire de représenter les différents intervenants pour chaque activité,
- Les modèles de tâches définissent plusieurs types d'enchaînement entre les tâches (parallèles, séquentielles, à déclenchement conditionnel etc.) et plusieurs types de tâches (critique, optionnelle, fréquente etc.) ; cette sémantique ne correspond pas aux diagrammes d'activités, dans lesquels les tâches peuvent être omises sur décision du chef de projet ou reprises plusieurs fois en fonction de la pertinence des choix effectués.

Pour conclure sur ce point, il nous semble plus opportun d'améliorer le système de navigation de l'aide en ligne et de préciser les produits entrants et sortants pour chaque activité que de traduire chaque diagramme d'activités, dont le formalisme lisible et peu complexe est accessible pour l'ensemble des rôles fonctionnels de la méthode Symphony.

Enfin, les participants ont à la fois constaté l'effectivité de la méthode, son efficience et ont déclaré leur intention de l'utiliser pour leurs propres projets. Bien qu'invalide du point de vue statistique (voir les limites de l'expérience, ci-dessous), ces premiers résultats suggèrent la justesse de notre approche. Néanmoins, de nombreux points négatifs ont été soulevés, sur lesquels nous avons concentré la suite de nos travaux.

8.2.3.1 Évolutions subséquentes à l'expérience

Cette première expérience nous a conduit à modifier plusieurs aspects de nos contributions, dont nous exposons quelques exemples ci-dessous.

Évolution du processus

Les descriptions des activités collaboratives ont été enrichies, chacune ayant été envisagée du point de vue des responsabilités des acteurs engagés dans l'activité et de l'artefact à produire. En particulier, les activités collaboratives ont été construites sous forme de coopérations, c'est-à-dire, d'activités impliquant les acteurs dans la réalisation d'un produit commun.

Évolution des modèles et langages

L'un des points essentiels abordé à la suite de l'expérience a été la réduction du nombre de modèles produits au cours du cycle de développement, selon différentes approches :

- la focalisation du développement sur les modèles utilisés dans le cadre des collaborations ou bien en tant que modèles communicationnels (notamment, les scenarii, les diagrammes d'activités UML, les prototypes) ;
- l'identification d'un squelette de produits essentiels pour la conception (c.f chapitre 4).

Évolution de l'outillage

Un nombre important de remarques a reflété l'incomplétude de l'aide en ligne, encore en cours de développement lors de l'expérience. Celles-ci nous ont donc permis de concentrer nos efforts sur la navigabilité de AGAP Lite (c.f chapitre 4).

8.2.3.2 Limites de l'expérience

Comme indiqué plus tôt, cette évaluation n'a pas de validité statistique : le nombre de sujets participant à l'expérience contraint à considérer cette dernière d'un point de vue qualitatif plutôt que quantitatif. Par ailleurs, on peut noter que le profil des sujets impliqués dans l'expérience, bien que proche en termes des compétences attendues, ne correspond pas exactement avec celui auquel la méthode est destinée, à savoir des concepteurs, analystes et chefs de projets issus de l'industrie.

Du point de vue du déroulement de la méthode, deux remarques méritent d'être formulées. Tout d'abord, le fait de n'avoir pas poursuivi l'expérience jusqu'à l'implémentation ne permet pas de garantir que les modèles produits lors des spécifications peuvent effectivement donner lieu à des systèmes de réalité mixte répondant de manière satisfaisante au cahier des charges. L'expérience sur l'effectivité des Objets Symphony, décrite à la section 8.3, fournit des éléments de réponse à ce problème. Ensuite, une évaluation exhaustive de la méthode aurait nécessité de comparer le développement à l'aide de Symphony avec d'autres cycles de développement plus classiques, tels que RUP ou Agile. La mise en œuvre d'une telle expérience aurait toutefois été longue et complexe, ce qui n'a pu être réalisé au cours de cette thèse.

8.3 Évaluation de l'effectivité de la méthode : implémentation des modèles d'analyse

La modélisation des Objets Symphony est centrale dans notre méthode :

- elle permet d'organiser les concepts essentiels du métier et de l'interaction,
- elle permet de définir un ensemble de conventions facilitant la systématisation du raffinement, des spécifications jusqu'à la conception,
- elle constitue un pivot pour la collaboration entre spécialistes du génie logiciel et spécialistes de l'interaction homme-machine.

L'évaluation de ces modèles est donc essentielle pour valider la construction de notre méthodologie. Plus particulièrement, il s'agit de s'assurer que celle-ci permet aux concepteurs de produire des applications constituées de composants réutilisables et intégrant une distinction claire des préoccupations entre l'interaction et le métier. L'une des hypothèses sur laquelle se basent nos contributions, et que nous évaluons ici, est que ces deux propriétés sont assurées grâce aux Objets Symphony.

8.3.1 Protocole expérimental

Afin de valider cette hypothèse, nous avons établi le protocole suivant : dans un premier temps, trois implémentations d'un fragment de la même application (l'application EDEMOI, présentée au chapitre 6 et destinée au visionnage de tests de sécurité aéroportuaire) sont réalisées, toutes trois à l'aide du langage Java, à partir des mêmes spécifications et modèles d'analyse. Considérant que ces derniers n'apposent pas de contrainte quant à la mise en œuvre des communications entre Objets Symphony, ce dernier paramètre constitue un point de variation permettant d'estimer la validité de notre hypothèse. Nous évaluons, par conséquent, le code d'implémentation de ces trois implémentations, à l'aide de métriques classiques du génie logiciel.

Cette évaluation doit nous permettre d'identifier les points d'évolution prioritaires de l'intergiciel Sonata. Dans le deuxième temps de l'expérimentation, nous réalisons les modifications répondant à ces points d'évolution prioritaires. Une nouvelle évaluation est ensuite réalisée sur l'implémentation complète de l'application EDEMOI, supportée par Sonata.

8.3.1.1 Comparaison de trois implémentations

Les trois implémentations comptent le même nombre d'Objets Symphony : deux Objets Interactionnels Entités, un Objet Interactionnel Processus, un Objet Métier Entité et un Objet Métier Processus. Les deux Objets Interactionnels Entités correspondent à deux facettes de la représentation de l'Objet Métier Entité. Par conséquent, une seule Translation est nécessaire pour connecter espace métier et espace d'interaction.

La première implémentation s'appuie sur l'une des premières versions de l'intergiciel Sonata, décrit au chapitre 6, dont on évalue le mécanisme de communication basé sur la programmation orientée aspects et les classes de Translation. L'utilisation de l'intergiciel permet de minimiser les redondances de code d'une part ; d'autre part, l'implémentation est organisée de façon à limiter la complexité des méthodes et classes, en systématisant le recours à des méthodes courtes et aux classes partie.

La deuxième implémentation, dite « Standard », utilise un modèle de communication plus classique, de type MVC [70]. Selon cette approche, l'Objet Interactionnel Processus prend le rôle de Contrôleur, chargé d'envoyer les requêtes idoines aux Objets Métier Entité (faisant office d'objets Modèle au sens MVC) et aux Objets Interactionnels Entité (faisant office d'objets Vue au sens MVC). Par conséquent, l'Objet Interactionnel Processus assume la responsabilité de la traduction des événements de l'espace interaction en appels de méthode dans l'espace métier. Le comportement correspondant à la Translation, défini au niveau Analyse, est donc réparti et intégré aux règles interactionnelles applicatives définies dans l'Objet Interactionnel Processus. Enfin, cette implémentation présente une politique « laxiste » quant à l'organisation des classes et méthodes : aucun effort particulier de factorisation du code ou de subdivision des méthodes n'a été effectué.

La troisième implémentation, dite « Optimisée », reprend l'approche de la précédente implémentation avec les variations suivantes : le modèle MVC est implémenté à l'aide d'un patron de communication proposé par Eckstein [49] permettant une totale indépendance entre les objets Modèle et Vue (et donc les Objets Métier Entité et les Objets Interactionnels Entité). Dans cette dernière implémentation, un soin particulier a été apporté à la structuration du code (factorisation du code redondant et subdivision des méthodes), afin de garantir une complexité minimale sur l'ensemble de l'application.

Deux programmeurs ont été sollicités pour construire ces systèmes, ayant des expériences en programmation orientée objet différentes. Le tableau 8.1 reprend les variables de ces évaluations.

Par la suite, nous avons estimé la qualité du code produit à l'aide de métriques logicielles. Celles-ci sont décrites dans la sous-section suivante.

8.3.1.2 Choix des métriques logicielles

Les propriétés de réutilisabilité et de distinction entre métier et interaction correspondent, en termes d'architecture logicielle, aux caractéristiques suivantes :

- les éléments du système doivent être faiblement dépendants les uns avec les autres ;
- les éléments du système susceptibles d'évoluer doivent être distincts et indépendants des éléments plus stables ;

TABLEAU 8.1 – Caractéristiques des trois implémentations de l'évaluation

	Expérience du programmeur	Communication entre Objets Symphony	Logique d'organisation des Objets Symphony	Taille (SLOC) [a]
Sonata	Expérimenté	Translations & aspects	Taille minimale	2 536
Standard	Intermédiaire	MVC avec Observateur	Taille indifférente	1 965
Optimisée	Intermédiaire	MVC selon Eckstein	Taille minimale	2 539

a. Source Lines Of Code – Lignes de code source

- afin d'être maintenables, les éléments du système doivent être de faible complexité algorithmique ; le cas échéant, il est préférable de découper un élément complexe en entités plus simples.

Le tableau 8.2 établi une correspondance entre ces propriétés et les types de métriques du génie logiciel permettant de les évaluer.

TABLEAU 8.2 – Propriétés et types de métriques associées

Propriété	**Métrique(s) associée(s)**
Réutilisabilité des Objets Symphony	Métriques de couplage & métriques d'instabilité
Découplage entre espaces métier & interactionnel	Métriques de couplage
Faible complexité	Métriques de complexité

L'estimation du couplage (afférent et efférent) entre composants, de leur instabilité (en fait le triptyque instabilité/abstraction/distance à la séquence principale proposé par Robert C. Martin [75]) ainsi que de leur complexité (cyclométrique) ont constitué trois catégories de mesures, pour lesquelles nous avons choisi des métriques représentatives de nos préoccupations. Nous précisons que la mesure de la complexité nous semble indicative, à besoins fonctionnels égaux, de la complexité des solutions techniques mises en œuvre pour permettre le découplage et la communication entre composants.

Nous recourons pour nos mesures à la notion de « catégorie de classes », ou « catégorie », au sens proposé par Booch [15] [1]. Dans nos différentes implémentations, les Objets Symphony correspondent à des catégories, les classes les composant étant fortement interdépendantes. Outre les mesures de complexité, nous effectuons les évaluations à ce niveau de granularité, concrètement parlant celui du paquetage Java. Les classes techniques ont, pour chaque implémentation, été structurées de façon à respecter l'équivalence entre catégorie de classes et paquetage Java. D'autre part, les appels aux bibliothèques incluses systématiquement

1. *« [Une catégorie de classes] est un ensemble logique de classes, dont certaines sont visibles depuis d'autres catégories de classes, d'autres cachées. Les classes d'une catégorie de classes collaborent pour fournir un service. »*

dans la machine virtuelle du langage (dont le nom est typiquement préfixé par `java.lang`) ne sont pas comptés, celles-ci étant particulièrement stables.

Nous décrivons ci-dessous les métriques de couplage, d'instabilité et de complexité, retenues pour notre évaluation.

Couplage afférent (Ca)

Le couplage afférent estime la responsabilité d'une entité dans la stabilité d'un système, c'est-à-dire le nombre d'entités dépendantes de l'entité considérée. Plus ce nombre est élevé, plus l'impact d'une modification sur cet élément sera important. La règle veut donc que les entités potentiellement réutilisables d'un système soient les plus stables possibles.

Couplage efférent (Ce)

À l'inverse du couplage afférent, le couplage efférent mesure la dépendance d'une entité vis-à-vis des autres éléments du système. Un module ayant un couplage efférent élevé sera donc particulièrement exposé à toute évolution du système et pourra être considéré comme instable. Il est donc préférable de concentrer le couplage efférent dans les points du système dont on sait qu'ils ne seront pas réutilisés, tels que les descriptions de règles applicatives.

Instabilité (I)

L'instabilité évalue la fragilité qu'une catégorie génère au sein d'un système, en tant qu'entité à la fois requise par d'autres catégories et dépendante d'entités. Cette mesure produit des valeurs sur l'intervalle $[0; 1]$, les valeurs proches de 0 ou 1 étant considérées comme « sûres ».

$$I = \frac{Ce}{(Ce + Ca)}$$

Abstraction (A)

L'abstraction d'une catégorie correspond à la proportion de classes abstraites qu'elle intègre. Cette mesure produit également des valeurs sur l'intervalle $[0; 1]$, 0 indiquant une catégorie concrète et 1 une catégorie complètement abstraite. Afin d'adapter la définition de l'abstraction à nos travaux, nous considérons qu'une classe est dite abstraite lorsqu'elle ne fait appel qu'aux primitives et classes de base du langage. Au contraire, une classe dite concrète fait appel à des classes externes au périmètre originel du langage, telles les bibliothèques graphiques.

$$A = \frac{\sum Classes_{abstraites}}{\sum Classes_{abstraites} + \sum Classes_{concrètes}}$$

Distance à la séquence principale (DMS)

Les travaux de Martin [75] suggèrent que les catégories de classes stable et abstraites, ou bien instables et concrètes, sont à favoriser au sein d'un système. De plus, il propose de

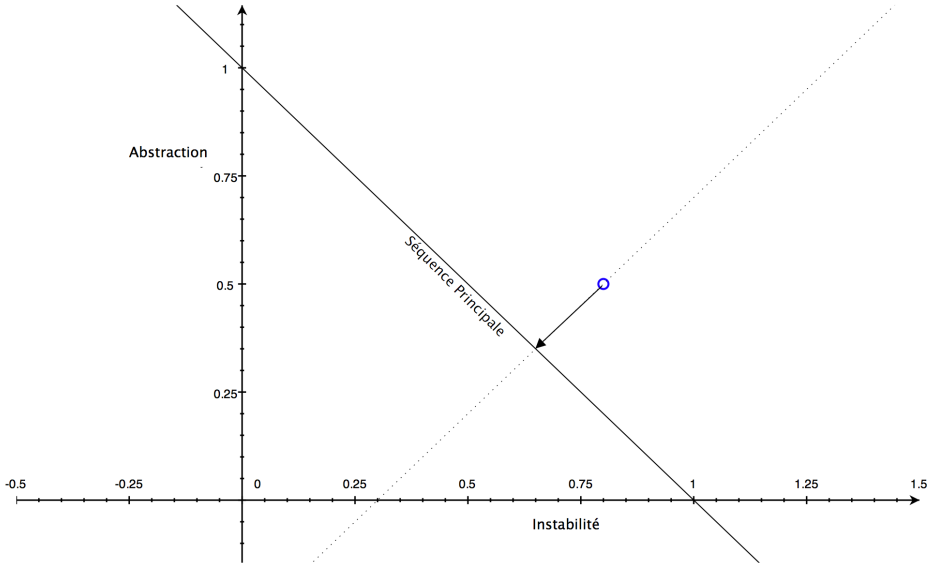


FIGURE 8.2 – Représentation de la distance à la séquence principale

considérer l'ensemble des valeurs intermédiaires reliant ces deux extrêmes – la *« séquence principale »* – comme acceptables. La figure 8.2 correspond à la représentation décrite par Martin.

Le couple $\{Instabilité, Abstraction\}$ caractérise donc les catégories d'un système, et la distance entre ce couple et sa projection orthogonale sur la séquence principale constitue la distance à la séquence principale (DMS – *Distance from the Main Sequence*). Les valeurs normalisées fournies par cette mesure s'inscrivent sur l'intervalle $[0; 1]$, les valeurs proches de 1 signalant une catégorie sensible au changement et peu réutilisable.

$$\text{pour tout } P = \{X, Y\}, \text{il existe } P_\perp = \{X_\perp, Y_\perp\}$$

$$\text{tel que } X_\perp = \frac{1-(Y-X)}{2}, Y_\perp = \frac{1+(Y-X)}{2}$$

$$\text{et } DMS = \sqrt{\frac{(X-X_\perp)^2+(Y-Y_\perp)^2}{2}}$$

Complexité cyclométrique (CC)

Cette métrique, proposée par McCabe [78], propose de mesurer la complexité d'un programme suivant le nombre de chemins algorithmiques, c'est-à-dire le nombre d'embranchements décisionnels, qu'il contient. Elle peut être maîtrisée en limitant d'une part la taille des méthodes de chaque classe, d'autre part le nombre d'embranchements décisionnels par méthode. Cette métrique n'est donc pas une incitation à simplifier le comportement d'un système, mais à maîtriser la complexité de ses fonctionnalités.

Les seuils suivants ont été définis par McCabe :

- $1 \leq CC \leq 4$ – Complexité faible ;
- $4 < CC \leq 7$ – Complexité modérée ;
- $7 < CC \leq 10$ – Complexité élevée ;
- $10 < CC$ – Complexité très élevée.

8.3.1.3 Hypothèses formulées sur les mesures des métriques

Considérant les différentes métriques présentées ci-dessus, nous avons formulé des hypothèses quant aux valeurs vérifiant les propriétés exprimées en section 8.3.1.2, pour chaque type de métrique. Celles-ci sont recensées dans le tableau 8.3.

Tableau 8.3 – Choix des métriques GL

	Nom	Hypothèse
Métriques de couplage	Couplage afférent (Ca) Couplage efférent (Ce)	Les couplages afférent et efférent entre deux paquetages Java correspondant l'un à un Objet Métier, l'autre à un Objet Interactionnel, est nul.
Métriques d'instabilité	Instabilité (I) Abstraction (A) Distance à la séquence principale (DMS)	Les Objets Symphony sont stables (instabilité proche de 0 ou 1) et abstraits (proportion de classes concrètes proche de 0), et les classes techniques sont concrètes.
Métriques de complexité	Complexité cyclométrique (CC)	Pour l'ensemble du projet, en particulier les Objets Entité : $1 \leq CC \leq 4$

8.3.2 Résultats

Les résultats des mesures réalisées sur les trois implémentations du fragment d'application EDEMOI sont décrits dans le tableau 8.4. Notons que la mesure de la distance à la séquence principale réalisée pour cette évaluation n'a pas été retenue, la définition de classe concrète utilisée pour cette mesure étant celle d'une classe ne contenant pas de méthodes abstraites. Dans le contexte des applications structurées à l'aide d'Objets Symphony (c'est-à-dire, utilisant dans un même paquetage des classes abstraites et des classes concrètes), cette définition ne fait pas sens.

Tableau 8.4 – Résultats des mesures sur les trois implémentations

	Couplage		Complexité Cyclométrique		Instabilité
	Ce_{moy}	Ce_{max}	CC_{moy}	CC_{max}	
Sonata	1.00	2	1.41	11	0.40
Standard	3.00	7	1.79	23	0.48
Optimisée	1.80	5	1.46	5	0.60

Nous avons limité l'évaluation du couplage au couplage efférent, considérant que le couplage afférent peut être déduit du premier et de la valeur d'instabilité. Concernant le couplage efférent, les trois implémentations présentent des valeurs moyennes et maximales basses.

Les trois implémentations présentent également des complexités cyclométriques moyennes faibles (entre 1.46 et 1.79). Les valeurs maximales de complexité cyclométrique des implémentations « Sonata » et « Optimisée » présentent des valeurs acceptables, la valeur de l'implémentation « Optimisée » témoignant d'une forte homogénéité.

La moyenne de l'instabilité étant peu éloquente, nous en considérons les valeurs en distinguant Objets Entités et Objets Processus. Les premiers présentent une instabilité nulle (aucun lien entrant), tandis que les seconds génèrent des valeurs proches de 1 (supérieures à 0.75) dénotant une majorité de dépendances sortantes.

La première évaluation nous a également laissé constater que la version de Sonata utilisée pour l'expérience imposent d'implémenter des classes utilitaires très similaires. À la suite de ces premiers résultats, nous avons réalisé plusieurs itérations d'amélioration de l'intergiciel Sonata, dont la dernière version est présentée au chapitre 6. Par exemple, conformément au modèle JavaBeans présenté au chapitre 6, chaque Objet Symphony est doté, dans cette version, d'une classe fabrique réalisant l'initialisation des instances et la connexion entre les instances et les classes techniques.

Suivant ces premières mesures sur des implémentations partielles de l'application EDEMOI, l'intergiciel Sonata a subit plusieurs itérations visant à faciliter son utilisation et à réduire notablement le volume de code d'implémentation « utilitaire » requis pour intégrer ce dernier avec les applications. Par exemple, la nouvelle version de l'intergiciel supprime la nécessité de décrire des classes fabriques (plusieurs dizaines de lignes de code), dont le comportement est factorisé dans une classe `AbstractEntityFactory` gérée par l'intergiciel (c.f chapitre 6).

De nouvelles mesures ont été réalisées, comme prévu, sur l'application EDEMOI complète (3 411 lignes de code source). Les résultats de ces mesures sont présentés au tableau 8.5.

TABLEAU 8.5 – Résultats des mesures pour l'application EDEMOI complète

Couplage		Complexité Cyclométrique		Instabilité
Ce_{moy}	Ce_{max}	CC_{moy}	CC_{max}	
3.05	15	1.69	16	0.59

La valeur moyenne de couplage efférent pour l'application EDEMOI est de 3.05 (2.66 en ne considérant que les Objets Symphony). On constate donc un couplage efférent plus élevé que lors de la première évaluation. Une étude approfondie révèle les caractéristiques suivantes :

- les valeurs de couplage (afférent et efférent) entre Objets Métier et Objets Interactionnels sont nulles, sur l'ensemble du projet ;
- les valeurs de couplage (afférent et efférent) entre Objets Symphony et classes technique est nulle, sur l'ensemble du projet ;
- les valeurs de couplage (afférent et efférent) les plus élevées sont réparties de façon homogène entre Objets Entité et Objets Processus.

De plus, bien que la taille actuelle de l'application EDEMOI soit plus importante de 34% que

le fragment évalué précédemment, la complexité du projet est constante.

Considérant nos hypothèses quant aux métriques d'abstraction et de distance à la séquence principale (DMS) utilisées dans ce manuscrit, les Objets Symphony (excluant donc les classes techniques) doivent idéalement avoir une instabilité proche et une abstraction proches de 1 (c.f figure 8.2).

Nos mesures indiquent que 93.5% des classes utilisées dans les Objets Symphony présentent s'inscrivent dans l'intervalle de valeurs d'instabilité $[0.9; 1.0]$. En étendant cet intervalle à $[0.85; 1.0]$, 100% des classes utilisées dans les Objets Symphony présentent des valeurs d'instabilité satisfaisantes (les valeurs de DMS des Objets Symphony pouvant être considérées comme basses voire faibles). Par ailleurs, tous les Objets Symphony ont une valeur d'abstraction égale à 1.

8.3.3 Discussion

Deux conclusions peuvent être tirées de l'expérience présentée ici : d'une part, malgré des expériences de développement différentes, les différentes implémentations du fragment d'application EDEMOI satisfont globalement nos hypothèses quant à la réutilisabilité et la modularité des applications développées à l'aide de la méthode Symphony.

D'autre part, les deux temps de l'expérimentation nous ont permis de développer un intergiciel efficace, confirmant les hypothèses formulées quant aux valeurs des mesures des métriques logicielles, pour des applications développées avec la méthode Symphony et l'intergiciel Sonata :

- le couplage entre Objets Métier et Objets Interactionnels est effectivement nul ;
- les Objets Symphony présentent une forte stabilité et une forte abstraction ;
- l'application EDEMOI est caractérisée par une complexité moyenne constante.

Concrètement, les Objets Symphony présentent les dépendances suivantes :

- dépendance aux données de référence ;
- dépendance à des Objets Entités (matérialisés lors de la spécification par les relations « utilise ») ;
- dépendance (ponctuelle) à certaines classes ciblées de l'intergiciel ;
- dépendance aux classes utilitaires du langage Java.

8.3.3.1 Limites de l'expérience

Du point de vue méthodologique, il nous semble nécessaire de multiplier les évaluations sur différents projets s'appuyant sur les modèles d'analyse de Symphony et/ou sur Sonata, afin d'apporter une considération quantitative à l'évaluation de nos hypothèses. De plus, une évaluation exhaustive des modèles d'analyse et de l'intergiciel Sonata nécessiterait d'une part de réaliser des comparaisons avec des implémentations de l'application EDEMOI ne suivant pas la méthode Symphony, d'autre part de comparer les valeurs mesurées avec celles provenant d'applications interactives tierces.

Du point de vue des résultats obtenus, il nous semble nécessaire de considérer avec prudence les valeurs d'instabilité relevées sur l'application EDEMOI. En effet, cette dernière présente une particularité structurelle, identifiée tardivement : tous les Objets Entités sont directe-

ment reliés à leur Objet Processus. Cette particularité favorise naturellement une répartition du couplage entre Objets Entités et Objets Processus de telle façon que les Objets Entités présentent une valeur de couplage efférent faible (ils n'utilisent pas d'autre Objet Entité), et les Objets Processus une valeur de couplage efférent plus forte (ils utilisent l'ensemble des Objets Entités du domaine conceptuel).

8.3.3.2 Évolutions et perspectives subséquentes à l'expérience

Le protocole expérimental de l'expérience a permis de réaliser plusieurs itérations corrigeant les défauts manifestes de l'intergiciel. Considérant les valeurs mesurées sur la dernière version de Sonata, selon les différentes métriques logicielles, nous considérons que les évolutions les plus pertinentes ne concernent désormais plus l'implémentation de l'intergiciel, mais sa mise en œuvre.

Cette expérience a également permis de recueillir informellement l'opinion du programmeur impliqué dans l'implémentation des versions « Standard » et « Optimisée », quant à la lisibilité et la facilité d'utilisation des modèles produits par la méthode Symphony. À l'instar de l'évaluation de l'efficience des phases amont de la méthode Symphony, il nous semble nécessaire d'approfondir l'efficience des modèles de niveau analyse.

De même, il serait pertinent d'évaluer la facilité d'utilisation de l'intergiciel Sonata, ce dernier n'ayant été utilisé que par l'auteur. À cette fin, les sources de l'intergiciel, sa documentation, et les tests unitaires et d'intégration qui lui sont associés sont proposés en open-source. Nous espérons ainsi évaluer l'efficience de l'intergiciel selon une dynamique double, consistant en des évaluations en laboratoire d'une part, et d'autre part en une validation par la communauté des développeurs Java.

8.4 Synthèse

Nous avons exposé dans ce dernier chapitre les évaluations réalisées sur deux aspects de nos contributions : l'efficience des collaborations entre des utilisateurs de la méthode issus de cultures informatiques différentes, et l'effectivité des modèles d'analyse et de l'intergiciel Sonata, pour l'implémentation d'applications réutilisables et modulaires, suivant la méthode Symphony.

Concernant la première évaluation, l'expérience a produit des résultats encourageants quant à la facilité d'utilisation de la méthode, telle que perçue par les utilisateurs. Bien que l'expérience ne puisse prétendre à des résultats quantitatifs significatifs, elle nous a incité à envisager plusieurs évolutions de la méthode Symphony et de l'outil AGAP Lite, qui ne demandent qu'à être de nouveau évalués à plus large échelle.

Les résultats de la seconde évaluation tendent également à conforter nos hypothèses quant à l'effectivité des modèles d'analyse et de l'intergiciel Sonata. Une nouvelle fois, l'absence de dimension quantitative de ces résultats nous interdit toutefois d'apporter une conclusion définitive quant à nos contributions.

De plus, notons que ces évaluations ne concernent que des fragments de la méthode Symphony étendue. Il serait notamment pertinent de vérifier l'efficience de la méthode auprès d'utilisateurs non experts, notamment en ce qui concerne les interactions en réalité mixte :

confrontés à notre méthode, et considérant un contexte d'interaction s'y prêtant (situation de mobilité, par exemple), de tels utilisateurs seraient-ils incités à se détourner des interfaces en formulaires pour envisager une interaction plus adaptée ?

Conclusion

Rappel de la problématique

Nos travaux concilient deux approches de conception apparemment antinomiques : celle des systèmes de réalité mixte, et celle des applications d'entreprise, qui s'appuient généralement sur un système d'information plus large. Les domaines de l'ingénierie de l'interaction homme-machine et du génie logiciel ont en effet élaboré des pratiques et une expertise de développement en relative isolation l'un de l'autre.

Le premier domaine a élaboré des processus, modèles et outils centrés sur la construction d'interfaces utilisables, mais sans prendre en compte les problématiques fonctionnelle et métier des systèmes. Le second domaine a, quant à lui, focalisé ses contributions sur les fonctionnalités et l'intégration dans les systèmes d'information, sans considérer l'utilisabilité des solutions.

Par ailleurs, bien que pertinents pour de nombreux contextes d'interaction, l'intégration d'une interaction mixte dans un écosystème d'entreprise constitue un risque multiple (financier, ergonomique, d'adoption par les utilisateurs) qu'il est indispensable de maîtriser.

L'état de l'art des méthodes intégrant les pratiques GL et IHM a démontré la rareté des solutions valorisant au même titre les fonctionnalités du système et son utilisabilité, voire l'absence de proposition concernant l'intégration des interactions post-WIMP dans les méthodes de développement issues du génie logiciel.

Spécifités de notre approche

Malgré l'absence de solution satisfaisante pour l'intégration des pratiques du GL et de l'IHM, l'analyse de l'état de l'art nous a permis d'identifier un certain nombre de principes répondant à nos objectifs de recherche. Nous avons ainsi constaté la prééminence de la modélisation par les cas d'utilisation, les scenarii, les diagrammes d'activités UML et les prototypes pour la communication avec les utilisateurs et les décideurs. Au sein des équipes de développement, scenarii et diagrammes d'activités sont également reconnus comme de bons supports pour des activités de conception communes. Les processus collaboratifs et les responsabilités des acteurs impliqués, toutefois, ne sont décrits qu'exceptionnellement.

Nous avons traité cette problématique en introduisant la notion d'activité collaborative dans notre méthode. Parmi les différents formes que peuvent prendre ces activités, la méthode Symphony étendue se limite à des coopérations, activités dont l'objectif est commun à ses acteurs, et produisant un modèle consensuel.

L'ensemble des modèles essentiels de la méthode, c'est-à-dire ceux utilisés soit pour la communication avec les utilisateurs et les décideurs, soit pour la coopération entre acteurs du développement, compose un squelette de modèles minimaux pour la conception. Dans le contexte d'une instance du cycle de développement plus légère, les activités de la méthode peuvent ainsi être réduites aux seules activités produisant ces modèles minimaux.

De plus, l'organisation des activités et produits de la méthode est basée sur une logique de production et consommation d'artefacts, ininterrompue de l'étude préalable jusqu'à l'implémentation. La transitivité des liens de traçabilité définis entre produits relie ainsi tout produit de la méthode à un besoin exprimé lors de l'étude préalable. Cette approche renforce le cadre la cohérence globale du cycle de développement, facilite son pilotage et donc la maîtrise des risques associés.

AGAP Lite, un outil pour la documentation de la méthode alternatif à l'outil AGAP de l'équipe SIGMA du LIG, offre une interface intuitive de navigation parmi l'enchaînement des activités et produits de la méthode.

Les modèles consensuels nous permettent, également, d'inscrire la spécification de l'IHM dans un sous-processus parallèle aux activités organisationnelles spécifiques du domaine métier. Les spécialistes de l'IHM et du GL ont ainsi toute latitude pour exprimer leurs compétences propres. La méthode proposée capitalise les savoir-faire existants en matière de conception d'interfaces homme-machine complexes et facilite leur développement. Les concepteurs sont ainsi encouragés à élaborer des systèmes de réalité mixte. À l'issue de ces activités parallèles, les spécifications du métier et de l'interaction sont organisées sous forme d'Objets Symphony.

Les Objets Symphony, point d'articulation des processus GL et IHM

La modélisation des concepts métier essentiels sous la forme d'Objets Métier est héritée de la méthode Symphony originelle. Nous avons étendu cette approche en proposant de représenter, selon le même formalisme, les concepts principaux de l'interaction homme-machine, appelés **Objets Interactionnels**. Ceux-ci permettent ainsi aux spécialistes GL et IHM de synchroniser leurs spécifications des besoins, réalisées indépendamment l'une de l'autre.

Réunis en un modèle global des concepts du métier et de l'interaction, les Objets Symphony donnent lieu à une structuration homogène des espaces métier et interactionnel facilitant la réalisation des phases d'analyse et de conception. Cette organisation régulière des Objets Symphony nous a incité à factoriser fortement leur conception et leur implémentation, dont de nombreux aspects (configuration, connexions et découplage entre Objets Métier et Objets Interactionnels) sont désormais pris en charge par l'intergiciel Sonata, conçu à cet effet.

En proposant des conventions de conception peu contraignantes, la description d'Objets Métier et Interactionnels permet ainsi de construire des applications conçues comme des ensembles de composants dont nous favorisons la réutilisabilité et la traçabilité.

Évaluations et perspectives

Les phases de spécification conceptuelle et de spécification organisationnelle et interactionnelle des besoins ont fait l'objet d'une évaluation qualitative, focalisée sur l'efficience de la méthode et sur l'utilisabilité de l'outil AGAP Lite. Malgré quelques défauts relevés quant à la navigabilité de l'outil de documentation AGAP Lite, corrigés depuis, cette première expérience a révélé des résultats encourageants. Néanmoins, le processus mérite d'être approfondi : en particulier, les résultats de l'expérience suggèrent de détailler les activités coopératives et de fournir des guides pour la résolution des responsabilités et décisions dans le cadre de ces activités.

Une seconde évaluation a été menée, portant sur le code source de plusieurs applications produites suivant la méthode, l'une d'elles utilisant l'intergiciel Sonata. Les résultats de ces évaluations tendent à montrer que le respect des conventions et de l'organisation des Objets Symphony permet d'obtenir des composants logiciels fortement découplés, avec d'une part l'implémentation d'une logique métier abstraite (dont les espaces métier et interactionnel sont eux-mêmes découplés), et d'autre part des composants techniques concrets. Considérant une logique métier constante d'une implémentation à l'autre, l'utilisation de Sonata permet de réduire la complexité des solutions produites.

La méthode Symphony étendue, ainsi que les outils AGAP Lite et Sonata, sont matures et accessibles sur internet, en open source. À ce jour, plusieurs applications ont été développées à l'aide de la méthode et des outils, dans un contexte académique.

Cependant, à présent que la spécification de la méthode Symphony étendue acquiert une forme de stabilité, il est urgent de la confronter à des développements en entreprise, sur des projets concrets. Nul doute qu'une telle démarche ferait surgir de nouvelles problématiques, dont l'approfondissement ne pourrait que bénéficier à notre méthode.

Il serait également nécessaire d'approfondir la gestion de la traçabilité dans la méthode Symphony. Considérant que l'outil AGAP Lite décrit l'ensemble des activités, relations entre activités et produits de la conception, nous pourrions étendre cet outil pour définir des règles de traçabilité, selon un processus systématique. En effet, les éléments méthodologiques sont tous reliés entre eux selon différentes relations de dépendance (utilisation, traçabilité, alternative…). Une instrumentation de ces graphes de relations pourraient ainsi permettre d'automatiser un large pan de la traçabilité des produits et des responsabilités.

Par ailleurs, la construction du sous-processus de spécification de l'IHM a démontré la faible variation des besoins méthodologiques entre la conception d'interfaces classiques (mais plus complexes que les formulaires) et la conception d'interfaces post-WIMP. La suite d'activités ainsi proposée semble à la fois adaptable à différents types d'interaction et suffisamment indépendante du processus général de conception pour être réutilisé dans d'autres contextes de développement, exclusivement orientés sur l'ingénierie de l'interaction homme-machine.

Les premières évaluations de la méthode Symphony étendue soulignent la difficulté d'estimer la qualité d'une méthode, pour deux raisons : d'une part, les nombreux facteurs à prendre en compte (contexte du développement, profil des acteurs impliqués dans la conception), d'autre part la diversité des composants (processus, modèles, langages, outils) à évaluer. Si les expériences présentées dans ce manuscrit ébauchent quelques éléments de réponse quant à la qualité de la méthode Symphony, il nous semble essentiel, à plus long terme,

d'étudier les conditions de généralisation de cette approche au domaine encore peu investi de l'évaluation des méthodes.

Nous évoquions en introduction à ce mémoire la vision de l'informatique intuitive et pervasive proposée par Mark Weiser. Incarnation exemplaire de cette conceptualisation, les multiples facettes des systèmes de réalité mixte sont chaque jour plus présentes dans notre quotidien. Nous espérons, en toute humilité, que nos travaux seront une contribution à cet élan désormais inexorable.

Annexes

ANNEXE A

Structure de l'intergiciel Sonata

Cette annexe détaille la structure de l'intergiciel Sonata, dont une représentation simplifiée est illustrée en figure A.1.

Si le concepteur d'applications Symphony manipule certaines classes de l'intergiciel (par exemple `AbstractEntityFactory`, `EntityObject`, `ConnectionTranslation`), d'autres restent invisibles à ce dernier. Nous considérons donc ci-dessous les interaction entre l'ensemble des classes de l'intergiciel.

Sonata instrumente les Objets Symphony tout au long de leur cycle de vie, selon trois phases :

1. L'enregistrement des Objets Symphony et des Connexions Symphony [1].
2. L'instanciation des Objets Symphony et l'établissement de Connexions Symphony.
3. L'interaction entre les objets source et cible des Connexions Symphony, via les classes Translation.

L'enregistrement des Objets Symphony est géré par la classe `AbstractEntityFactory` (méthode `register`). Cette opération est réalisée lors de l'initialisation de l'application : il s'agit de déclarer auprès de l'intergiciel les paquetages Java correspondant à des Objets Symphony. Les propriétés communes à toutes les instances d'un même Objet Symphony, ainsi que les noms des classes techniques utilisées par ces instances, sont également passées comme arguments de la méthode `register`.

L'enregistrement des Connexions Symphony est géré par la classe `Invoker`. Ces dernières sont décrites sous forme de fichiers XML ou JSON, et sont chargées par les spécialisations idoines de l'instance de la classe abstraite `InvokerDAO`, lequel encapsule les données de chaque Connexion Symphony dans des instances de la classe `BrokerReference`. Ces dernières permettent à l'`Invoker` de vérifier la conformité des requêtes adressées lors des événements de Connexion Symphony, par l'intermédiaire des aspects tissés sur les Objets Symphony.

Lors de l'instanciation d'un Objet Symphony, à l'aide de la méthode `createEntity`, l'instance de `AbstractEntityFactory` identifie la classe responsable de l'instanciation

1. Nous rappelons qu'une Connexion Symphony correspond au triplet $\{source, cible(s), Translation\}$, où $source$ et $cible(s)$ sont des Objets Symphony (Objets Métier et Objets Interactionnels).

(en se basant sur des conventions de nommage) et injecte dans l'instance nouvellement créée les propriétés définies lors de l'initialisation de l'application. De même, les classes techniques affectées à l'Objet Symphony sont instanciées et transmises à ce dernier.

Lorsqu'un aspect tissé sur un Objet Entité déclenche un événement de Connexion Symphony, l'`Invoker` crée une instance de la classe `Request`, destinée à contenir les détails de cette connexion, tels que le nom de la méthode de la classe Translation appelée, les arguments passés en paramètre, et l'Objet Entité responsable de l'événement de Connexion Symphony.

Une fois la requête complétée (dans l'aspect), elle est envoyée à l'`Invoker`, qui vérifie l'existence d'une Connexion Symphony (parmi les instances de `BrokerReference`) dont l'Objet Entité déclencheur de la requête serait la source. Le cas échéant, l'`Invoker` s'assure de l'existence d'une instance de Translation appropriée (c'est-à-dire, une classe héritant de la classe abstraite `ConnectionTranslation`), ou en crée une si nécessaire. Enfin, l'`Invoker` appelle la méthode décrite dans la référence, auprès de l'instance de Translation idoine.

La dernière étape de la création de Connexion Symphony concerne le comportement des méthodes exécutées depuis les instances de Translation, lors de la résolution d'une requête. Si, dans la pile d'exécution de la méthode, une instance d'Objet Entité est créée, l'`AbstractEntityFactory`, responsable de sa création, sollicite l'`Invoker` : s'il s'avère que l'Objet Entité créé dans ce contexte correspond à un objet cible de la Connexion Symphony en cours de résolution, celui-ci est associé à l'objet source et à la Translation. La Connexion Symphony est alors finalisée.

Nous rappelons au lecteur que le code source de l'intergiciel, accompagné de sa documentation, sont disponibles en open source à l'adresse :

`http://github.com/ggodet-bar/Sonata`

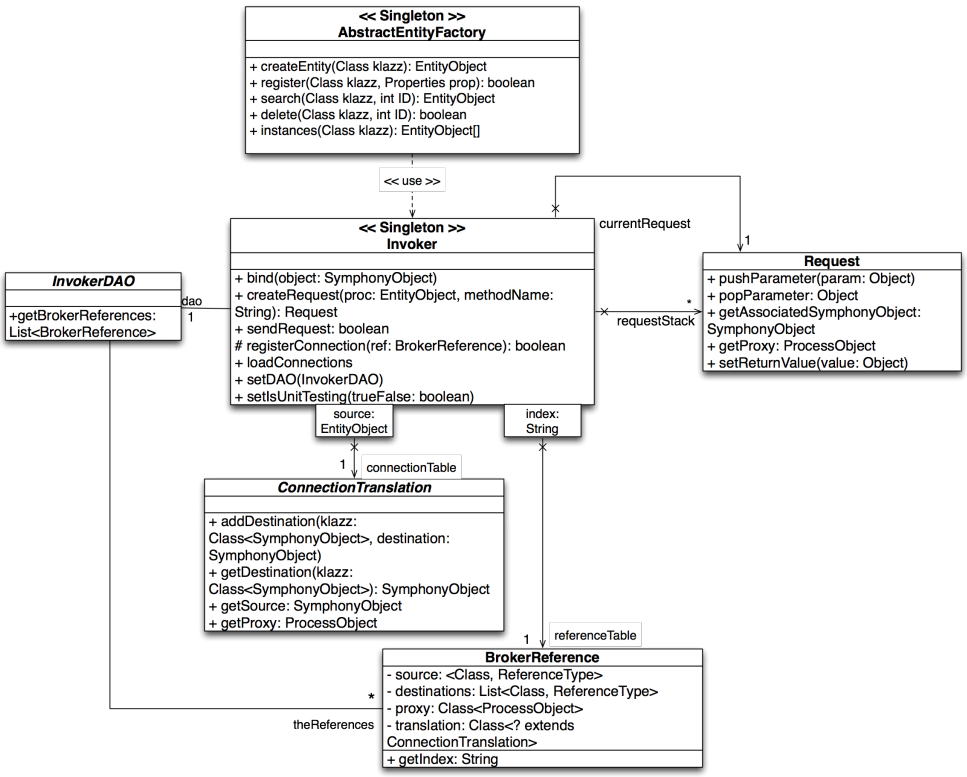


FIGURE A.1 – Diagrammes de classes simplifié de l'intergiciel Sonata

ANNEXE B

Exemple de règle de transformation de modèles

Nous exposons dans cette annexe un exemple de transformation implémentée à l'aide du langage ATL, correspondant à la génération d'un squelette de modèle d'analyse ne comportant que des Objets Métier, à partir d'un modèle d'Objets Métier de niveau spécification. Elle se rapproche de la transformation décrite textuellement au chapitre 7, page 183. Bien que se rapportant à un métamodèle incomplet et désormais obsolète, cette transformation rend compte des caractéristiques d'ATL, langage déclaratif, utilisant le concept de modèle comme entité de premier ordre. Les métamodèles et transformations sont à ce jour en cours de finalisation.

Une transformation ATL consiste, pour l'essentiel, en un en-tête, un ensemble de fonctions d'aide à l'implémentation (*helpers*) et un ensemble de règles.

L'en-tête déclare les métamodèles source et cible de la transformation (ici, le métamodèle des Objets Métier de niveau spécification – `BOSpec` –, et le métamodèle des Objets Métier de niveau analyse – `BOAnalysis`).

Les fonctions d'aide permettent généralement de factoriser les opérations de sélection et de filtrage sur les modèles. Par exemple, la fonction `getTarget` renvoie, pour une relation d'usage du modèle de spécification donnée, la classe « Interface » à laquelle elle doit être liée, dans le modèle d'analyse. On notera que la syntaxe de requête sur les modèles existants est, pour l'essentiel, similaire à celle utilisée dans le langage OCL. Les références aux éléments des métamodèles sont signalées par la syntaxe `Metamodel!MetamodelElement` (dans la transformation, `BOSpec!BOEntity` désigne ainsi l'élément `BOEntity` issu du métamodèle `BOSpec`, c'est-à-direle métamodèle des Objets Symphony de niveau spécification).

Cette transformation ATL s'appuie sur trois règles (signalées par le mot clef `rule`), décrivant la création d'un paquetage tripartite à partir d'un Objet Entité ou d'un Objet Processus (règles `BOEntity2Analysis` et `BOProcess2Analysis`), et la mise en relation via les dépendances « Utilise » (règle `BOUse2Analysis`).

```
1  module Spec2Analysis; -- Module Template
2  create OUT : BOAnalysis from IN : BOSpec;
3
4  ----------------------------------------
5  --------------HELPERS-------------------
6  ----------------------------------------
7
8  -- Retourne les Roles associes a un BOUse
9  -- Fonctionnement : on genere un Set avec
10 -- les noms des differents roles
11 -- nom role = nom du BOEntity/BOProcess source
12 -- + '_' + nom du BOEntity cible + '_Role'
13 -- On selectionne ensuite toutes les instances de Role
14 -- qui ont une correspondance
15 -- au niveau du nom dans le Set.
16 helper context BOSpec!BOUse def : getSources : Set( BOAnalysis!
       Role ) =
17   let sn : Set( String ) =
18     self.useSourceBOEntity->collect( e | e.name + '_' + self.
         useTarget.name + '_Role' )
19     ->union
20     (
21       self.useSourceBOProcess->collect( e | e.name + '_' + self
           .useTarget.name + '_Role' )
22     )
23   in
24     BOAnalysis!Role.allInstances( )->select( r | sn->includes(
         r.name ) );
25
26 ----------------------------------------
27
28 -- Retourne l'Interface associee a un BOUse
29 -- Fonctionnement : a partir des instances d'Interface, on
       selectionne la premiere a avoir une
30 -- correspondance au niveau du nom avec la cible du BOUse
31 helper context BOSpec!BOUse def : getTarget : BOAnalysis!
       Interface =
32   BOAnalysis!Interface->allInstances( )->flatten( )->select( e
       | e.name = self.useTarget.name )->asSequence( )->first( );
33
34
35 ----------------------------------------
36 ----------------REGLES------------------
37 ----------------------------------------
38
39 -- Transformation d'un BOEntity a son equivalent au niveau
       analyse.
40 -- Une Interface et un Master sont crees.
41 -- Autant de Role qu'il y a de reference vers une relation
       BOUse sont crees.
42 -- Les references entre les instances sont creees.
43 rule BOEntity2Analysis
44 {
45   from
46     e : BOSpec!BOEntity
47   to
48     ie : BOAnalysis!Interface
49       (
50         name <- e.name,
```

```
51          implementedBy <- me       -- reference au Master
52        ),
53      me : BOAnalysis!Master
54        (
55          name <- e.name + '_Master',
56          implements <- ie,       -- reference a l'Interface
57          collaboratesWith <- ree     -- reference a TOUS LES
        roles qui vont etre crees
58        ),
59      -- Pour chaque reference, creer un role
60      ree : distinct BOAnalysis!Role foreach( reee in e.useSource
        )
61        (
62          name <- e.name + '_' + reee.useTarget.name + '_Role',
63          collaboratesWith <- me
64        )
65  }
66
67  ------------------------------------------
68
69  rule BOProcess2Analysis
70  {
71    from
72      p : BOSpec!BOProcess
73    to
74      ip : BOAnalysis!Interface
75        (
76          name <- p.name,
77          implementedBy <- mp
78        ),
79      mp : BOAnalysis!Master
80        (
81          name <- p.name + '_Master',
82          implements <- ip,
83          collaboratesWith <- rep
84        ),
85      rep : distinct BOAnalysis!Role foreach( reep in p.useSource
        )
86        (
87          name <- p.name + '_' + reep.useTarget.name + '_Role',
88          collaboratesWith <- mp
89        )
90  }
91
92  ------------------------------------------
93
94  rule BOUse2Analysis
95  {
96    from
97      bu : BOSpec!BOUse
98    to
99      ou : BOAnalysis!Use
100     (
101       useSource <- bu.getSources,
102       useTarget <- bu.getTarget
103     )
104 }
105
106 ------------------------------------------
```

ANNEXE C

Synthèse des entretiens menés lors de l'expérience sur l'efficience de la méthode

Les tableaux ci-dessous recensent les opinions exprimées lors des deux entretiens avec les sujets impliqués dans l'évaluation de l'efficience de la méthode Symphony étendue. Les mentions GL et IHM entre parenthèses signalent le domaine d'expertise des sujets formulant l'avis.

TABLEAU C.1 – Points forts et points faibles de l'utilisation des Objets Interactionnels

	Points forts	Points faibles
Deuxième séance	• Visibilité des moyens d'interaction (2 IHM) • Relations entre les Objets Interactionnels (IHM) • Bon niveau de spécification (GL)	• Partiellement détaillé (GL) • Manquent des exemples d'Objets Interactionnels Processus (IHM)
Fin d'exercice	• La confrontation et le pont entre GL et IHM. Bon point de synchronisation (IHM)	• Il faut connaître la phase de conception pour voir l'utilité de ces objets (GL)

Tableau C.2 – Points forts et points faibles de l'utilisation des Objets Métier

	Points forts	Points faibles
Deuxième séance	• Encapsulation des concepts métier en modules (GL)	• Partiellement détaillé (GL)
Fin d'exercice	• Permet d'identifier les tâches à mener (IHM) • Utiles et intéressants pour bien adopter un point de vue utilisateur et non développeur (GL) • La confrontation et le pont entre GL et IHM. Bon point de synchronisation (IHM)	

Tableau C.3 – Points forts et points faibles du processus de développement

	Points forts	Points faibles
Deuxième séance	• Bon niveau de compréhension • Bon découpage selon les activités	• Le découpage en PMC parfois difficile (GL)
Fin d'exercice	• Des étapes pertinentes • La mise en commun au départ permet de dégager une vision et un langage commun. (IHM) • Facilite la mise en relation OI-OM (IHM) • Un travail indépendant entre GL et IHM (IHM) • Phase de choix de l'interaction très bonne étape, pourrait être plus détaillée avec des critères pour chaque solution. (IHM) • La définition par diagramme, qu'il faut enrichir pour chaque étape (GL)	• Manque une information sur le niveau de détails à atteindre

Tableau C.4 – Points forts et points faibles de la spécification (patrons processus)

	Points forts	Points faibles
Deuxième séance	• Utilisation des diagrammes d'activités dans chaque patron • Niveau de détail important • Présence d'exemples et de diagrammes • Aide au raffinement des idées • Pertinence des diagrammes de séquence et de la vision globale qu'ils apportent	• Manque d'exemples pour les novices • Manque un modèle pour les documents à produire • Manque le format des schémas à produire • Manquent des exemples pour les spécifications externes de l'interaction • Manque les diagrammes d'activité pour chaque étape • Manque une solution formelle, par exemple des diagrammes de classes
Fin d'exercice	• Un système de patrons détaillé • Des exemples pour chaque activité, pour accéder au niveau de détail • Suivi implicite par le texte • Vision globale de Symphony et de ses différentes phases	• Texte long à lire, privilégier des modèles • Manque de diagrammes

Tableau C.5 – Points forts et points faibles des enchaînements d'activités du processus

	Points forts	Points faibles
Deuxième séance	• Découpage pertinent, systématique • Activités logiquement liées entre elles • Cohérence et clarté des enchaînements • Pertinence du découpage en sous-activités	• Pas de vision claire des documents en entrée/sortie • Manque de guidage entre activités • Manque une vue globale de l'enchaînement des patrons
Fin d'exercice		• Manque une synthèse des entrées/sorties • Enchaînements laborieux, il manque un positionnement sur le graphe des patrons

Tableau C.6 – Points forts et points faibles du langage UML

	Points forts	Points faibles
Deuxième séance	• Modéliser l'organisation des processus métier et des concepts du domaines • Langage « universel »	• Découpage activités et actions • Trop de schémas • Difficulté à assurer la cohérence inter-diagrammes
Fin d'exercice		

Tableau C.7 – Points forts et points faibles du langage CTT

	Points forts	Points faibles
Deuxième séance	• [Permet de décrire l'] enchaînement des actions • Aide à la formalisation de la tâche de l'utilisateur • Bon niveau d'abstraction	
Fin d'exercice		• Pas d'outils satisfaisant pour l'édition, il manque des pointeurs sur les outils de représentation de modèles CTT • Niveau d'abstraction un peu haut • Difficulté d'estimer la complexité de ce qui est produit

Tableau C.8 – Points forts et points faibles du langage ASUR

	Points forts	Points faibles
Deuxième séance		• Notation non expliquée
Fin d'exercice	• Permet de voir le lien entre les objets et les utilisateurs • Haut niveau de détail dans les deux patrons [???] • Concepts simples, rapidité de prise en main pour un novice • au moins on n'a pas de petit nuage et quelques contraintes de cohérence [???]	• Difficulté à représenter un objet physique ET un dispositif d'entrée

Tableau C.9 – Points forts et points faibles l'outil d'aide en ligne AGAP Lite

	Points forts	Points faibles
Deuxième séance	• Bon support de la méthode • Simple, clair • Un guidage efficace pour le concepteur • Description des activités et des enchaînements • Navigation entre les patrons • Structure identique des patrons • Présence d'exemples et de diagrammes	• Problème de navigation et de vue d'ensemble • Erreurs d'ergonomie et de conception : *« l'organisation doit être ordonnée selon l'enchaînement des phases et des activités ; il manque un lien pour ne pas avoir à remonter la hiérarchie à chaque fin d'étape. Il manque une vision miniature de [la localisation] dans le Y et dans la phase, du type 'vous êtes ici' »* • [Liste des patrons] mal organisée • Pas de visibilité synthétique de la hiérarchie • Des problèmes d'ergonomie dans la navigation (IHM et GL) • Problème de l'aide pas actualisée au début de l'expérience
Fin d'exercice	• Structuration en rubriques • Niveau de détail des patrons • Facilité de repérage • Navigation par les phases	

Bibliographie

[1] Alexander, C., Ishikawa, S., and Silverstein, M. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, 1977.

[2] Arlow, J., and Neustadt, I. *UML and the Unified Process: Practical Object-Oriented Analysis & Design.* Addison-Wesley, 2002.

[3] Asuncion, H. U., François, F., and Taylor, R. N. An end-to-end industrial software traceability tool. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (New York, NY, USA, 2007), ACM, pp. 115–124.

[4] Balme, L., Demeure, A., Barralon, N., Coutaz, J., and Calvary, G. CAMELEON-RT: a software architecture reference model for distributed, migratable, and plastic user interfaces. In *Second European Symposium on Ambient Intelligence, EUSAI'04* (Pennsylvania, USA, octobre 2004), ACM Press, pp. 251–258.

[5] Barthet, M.-F. *Logiciels interactifs et ergonomie : modèles et méthodes de conception.* Dunod Informatique, 1988.

[6] Bass, L., John, B. E., Juristo, N., and Sanchez-Segura, M.-I. Usability-supporting architectural patterns. In *ICSE'04* (2004).

[7] Bastien, J. C., and Scapin, D. L. Ergonomic criteria for the evaluation of human-computer interfaces. Tech. Rep. RT-0156, INRIA, 06 1993.

[8] Beaudoin-Lafon, M. Instrumental interaction: An interaction model for designing post-WIMP user interfaces. *CHI'00, CHI Letters* (2000), 446–453.

[9] Beck, K. *Extreme Programming explained: Embrace change.* Addison-Wesley, 1999.

[10] Beck, K., and Cunningham, W. A laboratory for teaching object oriented thinking. *SIGPLAN Not. 24*, 10 (1989), 1–6.

[11] Bérard, F. The magic table: Computer-vision based augmentation of a whiteboard for creative meetings. In *IEEE Workshop on Projector-Camera Systems (PROCAM)* (2003).

[12] Berry, R. E. Common user access—a consistent and usable human-computer interface for the saa environments. *IBM Syst. J. 27*, 3 (1988), 281–300.

[13] Beyer, H., and Holtzblatt, K. Contextual design. *interactions 6*, 1 (1999), 32–42.

[14] Boehm, B. W. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes 11*, 4 (1986), 14–24.

[15] BOOCH, G. *Object-Oriented Analysis and Design with Applications (2nd Edition)*. Addison-Wesley Object Technology Series, 1994.

[16] BORCHERS, J. O. A pattern approach to interaction design. In *Designing interactive systems: Processes, practices, methods and techniques* (2000).

[17] CARD, S., MORAN, T. P., AND NEWELL, A. *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[18] CARROLL, J. M. Scenario-based design. In *Handbook of Human-Computer Interaction*, M. Helander, T. K. Landauer, and P. Prabhu, Eds. Elsevier Science B.V., 1997, pp. 383–406.

[19] CARROLL, J. M. Five reasons for scenario-based design. *Interacting with Computers 13*, 1 (2000), 43 – 60.

[20] CHALON, R. *Réalité Mixte et Travail Collaboratif : IRVO, un modèle de l'Interaction Homme-Machine*. Thèse de doctorat, École Centrale de Lyon, Lyon, décembre 2004.

[21] CHALON, R., AND DAVID, B. T. Modélisation de l'interaction collaborative dans les systèmes de réalité mixte. In *Actes de la 16eConférence francophone sur l'Interaction Homme-Machine IHM'04* (Namur, Belgique, 2004), pp. 37–44.

[22] CHALON, R., AND DAVID, B. T. IRVO: an Interaction Model for designing Collaborative Mixed Reality systems. *ArXiv e-prints 707* (juillet 2007).

[23] CHAN, F. K. Y., AND THONG, J. Y. L. Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems 46*, 4 (mars 2009), 803–814.

[24] CHARFI, S., DUBOIS, E., AND BASTIDE, R. Articulating interaction and task models for the design of advanced interactive systems, tamodia 2007. In *Proceedings of 6th Internation workshop on TAsk MOdels and DIAgrams* (2007), vol. 4849/2007, Springer-Verlag, pp. 70–83.

[25] CHARFI, S., SCAPIN, D. L., AND DUBOIS, E. Identification et prise en compte de propriétés ergonomiques pour la modélisation et la conception de sim. In *IHM '08: Proceedings of the 20th International Conference of the Association Francophone d'Interaction Homme-Machine* (New York, NY, USA, 2008), ACM, pp. 55–62.

[26] CHASTINE, J. W., NAGEL, K., ZHU, Y., AND YEARSOVICH, L. Understanding the design space of referencing in collaborative augmented reality environments. In *Proceedings of GI'07* (2007), ACM, Ed., pp. 207–214.

[27] COCKBURN, A. *Crystal Clear, A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, 2004.

[28] CONSTANTINE, L. Beyond user-centered design and user experience: Designing for user performance. *Cutter IT Journal 17*, 2 (mars 2004), 16–25.

[29] CONSTANTINE, L., BIDDLE, R., AND NOBLE, J. Usage-centered design and software engineering: Models for integration. In *Proceedings of the IFIP TC13 workshop on Closing the gaps: Software engineering and Human-Computer Interaction* (2003), M. B. Harning and J. Vanderdonckt, Eds.

[30] CONTE, A., FREDJ, M., GIRAUDIN, J.-P., AND RIEU, D. P-Sigma : un formalisme pour une représentation unifiée de patrons. In *Actes du 19econgrès Informatique des organisations et systèmes d'information et de décision INFORSID'2001* (Genêve, juin 2001).

[31] CONTE, A., GIRAUDIN, J.-P., HASSINE, I., AND RIEU, D. Un environnement et un formalisme pour la définition, la gestion et l'application de patrons. *Ingénierie des Systèmes d'Information (ISI) 6*, 2 (2002).

[32] COPLIEN, J. O. A generative development-process pattern language. 183–237.

[33] CORLETT, D. Design: innovating with ovid. *interactions 7*, 4 (2000), 19–26.

[34] COUTAZ, J. PAC, an object oriented model for dialog design. In *Proceedings of INTERACT'87* (1987), H.-J. Buillinger and B. Shackel, Eds.

[35] COUTAZ, J. *Modèles en Interaction Homme-Machine.* UFR-IMAG, Grenoble, 2005–2006.

[36] COUTRIX, C., AND NIGAY, L. Mixed reality: A model of mixed interaction. In *Proceedings of the 8th International Conference on Advanced Visual Interfaces AVI'2006* (Venezia, 2006), ACM Press, pp. 43–50.

[37] CUNNINGHAM, W., AND BECK, K. Constructing abstractions for object-oriented applications. Tech. Rep. CR-87-25, Computer Research Laboratory, Tektronix, Inc., 1987.

[38] DA SILVA, B. S., NETTO, O. A. M., AND BARBOSA, S. D. J. Promoting a separation of concerns via closely-related interaction and presentation models. In *CLIHC'05: Proceedings of the 2005 Latin American conference on Human-computer interaction* (New York, NY, USA, 2005), pp. 170–181.

[39] DAVID, B. T. IHM pour les collecticiels. In *Réseaux et systèmes répartis*, vol. 13. Hermès, novembre 2001, pp. 169–206.

[40] DAVIS, F. D. Perceived usefulness, perceived ease of use and user acceptance of information technology. *MIS Quarterly 13*, 3 (1989).

[41] DAYTON, T., MCFARLAND, A., AND KRAMER, J. *Bridging User Needs to Object Oriented GUI Prototype via Task Object Design.* CRC Press, 1998, ch. Bridging User Needs to Object Oriented GUI Prototype via Task Object Design, pp. 15–56.

[42] DE PAULA, M. G., BARBOSA, S. D. J., AND DE LUCENA, C. J. P. Conveying human-computer interaction concerns to software engineers through an interaction model. In *CLIHC'05: Proceedings of the 2005 Latin American conference on Human-computer interaction* (New York, NY, USA, 2005), pp. 109–119.

[43] DIDONET DEL FABRO, M., BEZIVIN, J., JOUAULT, F., AND VALDURIEZ, P. Applying generic model management to data mapping. In *Proceedings of the Journées Bases de Données Avancées (BDA 2005)* (Saint-Malo, France, octobre 2005).

[44] D'SOUZA, D. F., AND CAMERON WILLS, A. *Objects, Components and Frameworks with UML, the Catalysis Approach.* Addison-Wesley, 1998.

[45] DUBOIS, E. *Chirurgie Augmentée, un cas de Réalité Augmentée. Conception et réalisation centrées sur l'utilisateur.* Thèse de doctorat, Université Joseph Fourier, 2001.

[46] DUBOIS, E., GRAY, P. D., AND NIGAY, L. Asur++: A design notation for mobile mixed systems. *Interacting With Computers 15* (2003), 497–520.

[47] DUBOIS, E., SCAPIN, D., BACH, C., DUPUY, S., MANSOUX, B., TIGLI, J.-Y., AND VACHET, C. Méthodes et outils pour les systèmes mixtes. In *Atelier Méthodes et Outils pour les Systèmes Mixtes à IHM'04* (Namur, Belgique, août 2004).

[48] DUPUY-CHESSA, S., AND DUBOIS, E. Requirements and impacts of model-driven engineering on mixed systems design. In *IDM'05* (2005), pp. 43–54.

[49] ECKSTEIN, R. *Java SE Application Design With MVC.* Sun Microsystems Inc., 2007.

[50] FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. *Computer Graphics: Principles and Practice.* Addison-Wesley, 1990.

[51] FOX, D., SILLITO, J., AND MAURER, F. Agile methods and User-Centered Design: How these two methodologies are being successfully integrated in industry. In *Proceedings of Agile 2008* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 63–72.

[52] GAMMA, E., HELM, R., JOHNSON, R. E., AND VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[53] GIRAUDIN, J.-P. Exigences pour de nouvelles méthodes d'ingénierie des SI. In *Workshop OCM-SI* (Nantes, 2002).

[54] GODET-BAR, G., DUPUY-CHESSA, S., AND NIGAY, L. Towards a system of patterns for the design of multimodal interfaces. In *Proceedings of 6th International Conference on Computer-Aided Design of User Interfaces CADUI'2006* (Bucharest, juin 2006), G. Calvary, C. Pribeanu, G. Santucci, and J. Vanderdonckt, Eds., Springer-Verlag, pp. 27–40.

[55] GREBICI, K., BLANCO, E., AND RIEU, D. Toward non mature information management in collaborative design processes. In *Proceedings of the International Conference on Engineering Design ICED'05* (Melbourne, Australia, août 2005).

[56] HARTSON, H. R., SIOCHI, A. C., AND HIX, D. The UAN: A user-oriented representation for direct manipulation interface designs. *Transactions on Information Systems (TOIS) 8*, 3 (1990), 181–203.

[57] HASSINE, I. *Spécification et formalisation des démarches de développement à base de composants métier: la démarche Symphony.* Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, décembre 2005.

[58] HASSINE, I., RIEU, D., BOUNAAS, F., AND SEGHROUCHNI, O. Symphony: a conceptual model based on business components. In *SMC'02, IEEE International Conference on Systems, Man, and Cybernetics* (2002), vol. 2.

[59] HOSSEINI-KHAYAT, A., GHANAM, Y., PARK, S., AND MAURER, F. ActiveStory Enhanced: Low-fidelity prototyping and wizard of Oz usability testing tool. In *Proceeding of 10th International Conference on Agile Processes and eXtreme Programming in Software Engineering XP 2009* (2009), vol. 31, Springer Berlin Heidelberg, pp. 257–258.

[60] IUT2 DE GRENOBLE. *Conception de Systèmes à Objets – Cours OMGL-4*, 2008–2009.

[61] JACOBSON, I. Basic use case modeling. *Report on object analysis and design 1*, 2 (1994), 15–19.

[62] JACOBSON, I., BOOCH, G., AND RUMBAUGH, J. *The Unified Software Development Process.* Addison-Wesley, 1999.

[63] JAUSSERAN, E. Démarche Symphony étendue, formalisation et expérimentation sur un système d'information hospitalier. Mémoire d'ingénieur CNAM, Conservatoire National des Arts et Métiers de Grenoble, 2005.

[64] JOHN, B. E., BASS, L. J., GOLDEN, E., AND STOLL, P. A responsibility-based pattern language for usability-supporting architectural patterns. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems* (New York, NY, USA, 2009), ACM, pp. 3–12.

[65] JOUAULT, F., AND KURTEV, I. On the architectural alignment of ATL and QVT. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing* (New York, NY, USA, 2006), ACM, pp. 1188–1195.

[66] JURAS, D., DUPUY-CHESSA, S., AND RIEU, D. Vers une méthode de développement pour les systèmes mixtes. *Revue Génie Logiciel*, 77 (2006), 31–36.

[67] KETTANI, N., MIGNET, D., PARÉ, P., AND ROSENTHAL-SABROUX, C. *De Merise à UML*. Eyrolles, 2001.

[68] KICZALES, G., LAMPING, J., MENDHEKAR, A., MAEDA, C., LOPES, C., LOINGTIER, J., AND IRWIN, J. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP)* (1997), pp. 220–242.

[69] KIM, D.-K., AND WHITTLE, J. Generating uml models from domain patterns. In *SERA '05: Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 166–173.

[70] KRASNER, G., AND POPE, S. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming 1*, 3 (1988), 26–49.

[71] KROGSTIE, J. Integrating the understanding of quality in requirements specification and conceptual modeling. *SIGSOFT Softw. Eng. Notes 23*, 1 (1998), 86–91.

[72] KRUCHTEN, P. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2003.

[73] LANGE, C., AND CHAUDRON, M. Managing model quality in UML-based software development. In *Proceeedings of the 13th International Workshop on Software Technology and Engineering Practice (STEP'05)* (2005), pp. 7–16.

[74] LEMIEUX, F., AND DESMARAIS, M. C. Conception centrée sur l'utilisateur lors de la définition des exigences en RUP : Une étude de cas. *RSTI - ISI. Interaction homme-machine dans les SI 12* (décembre 2007), 9–38.

[75] MARTIN, R. C. *Object Oriented Design Quality Metrics: An Analysis of Dependencies*, vol. 2. ROAD, 1995.

[76] MARTIN, R. C. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.

[77] MATTHEWS, M., CHALLA, M., CHU, C., JIAN, G., SEICHTER, H., AND GRASSET, R. Evaluation of spatial abilities through tabletops AR. In *Proceedings of the 7th HCINZ* (Nouvelle-Zélande, 2007), ACM, Ed., pp. 17–24.

[78] MCCABE, T. A complexity measure. *IEEE Transactions on Software Engineering SE-2*, 4 (décembre 1976), 308–320.

[79] MOODY, D. L. The method evaluation model: A theoretical model for validating information systems design methods. In *Proceedings of 11th European Conference on Information Systems ECIS 2003* (2003).

[80] MORI, G., PATERNÒ, F., AND SANTORO, C. CTTE: Support for developing and analysing task models for interactive system design. In *IEEE Transactions on Software Engineering* (août 2002), IEEE Press, Ed., pp. 797–813.

[81] NIGAY, L., AND COUTAZ, J. Espaces conceptuels pour l'interaction multimédia et multimodale. In *TSI 96* (1996).

[82] NORMAN, D. A. Human-centered design considered harmful. *interactions 12*, 4 (2005), 14–19.

[83] NUNES, D. N. J. *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach.* Thèse de doctorat, Universidade de Madeira, 2001.

[84] NUNES, N. J., AND CUNHA, J. F. *Wisdom—Whitewater interactive system development with object models.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001, pp. 197–243.

[85] NUSEIBEH, B., AND EASTERBROOK, S. Requirements engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering* (New York, NY, USA, 2000), ACM, pp. 35–46.

[86] OBJECT MANAGEMENT GROUP. *Unified Modeling Language: Infrastructure (version 2.0/07-11-04)*, 2004.

[87] OBJECT MANAGEMENT GROUP. *Unified Modeling Language: Superstructure (version 2.0/05-07-04)*, 2004.

[88] OBJECT MANAGEMENT GROUP. *MOF 2.0/XMI Mapping Specification, V2.1.1*, 2007.

[89] OBJECT MANAGEMENT GROUP. *Unified Modeling Language: Superstructure (version 2.2/09-02-02).* Object Management Group, février 2009.

[90] OUSSALAH, M. C., HASSINE, I., RIEU, D., BOUNAAS, F., JAUSSERAN, E., FRONT, A., AND GIRAUDIN, J.-P. *Ingénierie des composants.* Vuibert informatique, 2005, ch. 4.

[91] OVIATT, S. *The Human-Computer Interaction Handbook.* Lawrence Erlbaum Associates, 2008, ch. Multimodal interfaces, pp. 413–432.

[92] PANET, G., AND LETOUCHE, R. *Merise/2 : Modèles et techniques Merise avancées.* Les Éditions d'Organisation, 1994.

[93] PATERNO, F. *The handbook of task analysis for human-computer interaction.* Lawrence Erlbaum Associates, 2003, ch. ConcurTaskTrees: An Engineered Notation for Task Models, pp. 483–503.

[94] PRESSMAN, R. *Software Engineering: A Practitioner's Approach.* McGraw-Hill Science/Engineering/Math, 2004.

[95] RENEVIER, P. *Systèmes Mixtes Collaboratifs sur Supports Mobiles : Conception et Réalisation.* Thèse de doctorat, Université de Grenoble, Grenoble, 2004.

[96] RIEU, D. *Ingénierie des Systèmes d'Information.* Mémoire d'habilitation à diriger les recherches, Institut National Polytechnique de Grenoble, 1999.

[97] ROBERTS, D. *Human-Centered Software Engineering - Integrating Usability*. Springer-Verlag, 2005, ch. Coping with complexity, pp. 201–217.

[98] ROYCE, W. W. Managing the development of large software systems. In *Proceedings of IEEE WESCON* (août 1970), TRW, pp. 328–338.

[99] SALBER, D. *De l'interaction individuelle aux systèmes multi-utilisateurs. L'exemple de la Communication Homme-Homme Médiatisée*. Thèse de doctorat, Université de Grenoble, Grenoble, 1995.

[100] SCHWABER, K. *Agile Project Management with Scrum*. Microsoft Press, 2004.

[101] SEBILLOTTE, S. Methodology guide to task analysis with the goal of extracting relevant characteristics for human-computer interfaces. *International Journal of Human-Computer Interaction 7*, 4 (1995), 341–363.

[102] SEFFAH, A., DESMARAIS, M. C., AND METZKER, E. HCI, usability and software engineering integration: Present and future. *Human-Centered Software Engineering — Integrating Usability in the Software Development Lifecycle 8* (2005), 37–57.

[103] SELECT BUSINESS SOLUTIONS, I. Select Perspective - an agile process v 1.2 (whitepaper). Tech. rep., Select Business Solutions, Inc., 2006.

[104] SELIGMANN, P., WIJERS, G. M., AND SOL, H. G. Analyzing the structure of IS methodologies: an alternative approach. In *First Conference on Information Systems* (1989).

[105] SINNIG, D., CHALIN, P., AND KHENDEK, F. Towards a common semantic foundation for use cases and task models. *Electronic Notes in Theoretical Computer Science 183* (2007), 73 – 88. Proceedings of the First International Workshop on Formal Methods for Interactive Systems (FMIS 2006).

[106] SOUSA, K., AND FURTADO, E. From usability tasks to usable user interfaces. In *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams* (New York, NY, USA, 2005), A. Dix and A. Dittmar, Eds., ACM Press, pp. 103–110.

[107] SOUSA, K., FURTADO, E., AND MENDONÇA, H. UPi: a software development process aiming at usability, productivity and integration. In *CLIHC '05: Proceedings of the 2005 Latin American conference on Human-computer interaction* (New York, NY, USA, 2005), ACM, pp. 76–87.

[108] SOUSA, K. S. *UPi – A Software Development Process Aiming at Usability, Productivity and Integration*. Thèse de doctorat, Universidade de Fortaleza, 2005.

[109] SOUSA, K. S., AND FURTADO, E. An approach to integrate hci and se in requirements engineering. In *Proceedings of the IFIP TC13 workshop on Closing the gaps: Software engineering and Human-Computer Interaction* (2003), M. B. Harning and J. Vanderdonckt, Eds.

[110] SOUSA, K. S., MENDONÇA, H., AND FURTADO, E. Applying a multi-criteria approach for the selection of usability patterns in the development of dtv applications. In *Simpósio sobre fatores humanos em sistemas computacionais, IHC 2006* (Brasil, 2006).

[111] STANDISH GROUP. The Standish Group report: CHAOS. Tech. rep., Standish Group, 1995.

[112] STANDISH GROUP. CHAOS: A recipe for success. Tech. rep., Standish Group, 1999.

[113] STANDISH GROUP. Extreme CHAOS. Tech. rep., Standish Group, 2001.

[114] SUTCLIFFE, A. Convergence or competition between software engineering and human computer interaction. *Human-Centered Software Engineering — Integrating Usability in the Software Development Lifecycle 8* (2005), 71–84.

[115] TARBY, J.-C. *The handbook of task analysis for human-computer interaction.* Lawrence Erlbaum Associates, 2003, ch. One Goal, Many Tasks, Many Devices: From Abstract User Task Specification to User Interfaces, pp. 531–550.

[116] TARBY, J.-C., AND BARTHET, M.-F. The DIANE+ method. In *Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces* (1996), pp. 95–120.

[117] TARBY, J.-C., AND BARTHET, M. F. Analyse et modélisation des tâches dans la conception des systèmes d'information : la méthode DIANE+. In *Analyse et conception de l'IHM*. Hermès, 2001.

[118] THEVENIN, D. *Adaptation en interaction homme-machine : le cas de la Plasticité*. Thèse de doctorat, Université Joseph Fourier, 2001.

[119] TIDWELL, J. *Designing Interfaces.* O'Reilly Media, 2005.

[120] TUCKER, A. B. *Computer Science Handbook, Second Edition.* Chapman & Hall/CRC, 2004.

[121] VAN WELIE, M., AND HALLVARD, T. Interaction patterns in user interfaces. In *PLoPoo* (2000).

[122] VERNIER, F. *La multimodalité en sortie et son application à la visualisation de grandes quantités d'information.* Thèse de doctorat, Université Joseph-Fourier, 2001.

[123] WEISER, M. The computer for the 21st century. *Scientific American* (septembre 1991), 94–104.

[124] WEISER, M. Some computer science issues in ubiquitous computing. *Communications of the ACM 36*, 7 (juillet 1993), 75–84.

Webographie

[Agi] *Manifesto for Agile Software Development.* Consulté le 28/09/2009.
http://agilemanifesto.org/.

[Art] Human Interface Technology Laboratory. *ARToolkit.* Consulté le 28/09/2009.
www.hitl.washington.edu/artoolkit/.

[Asp] The Eclipse Foundation. *The AspectJ Project.* Consulté le 27/09/2009.
http://www.eclipse.org/aspectj/.

[Bat] The Apache XML Graphics Project. *Batik.* Consulté le 04/10/2009.
http://xmlgraphics.apache.org/batik/.

[Dot] Microsoft. *Microsoft .NET Framework.* Consulté le 27/09/2009.
http://www.microsoft.com/NET/.

[Ecl] The Eclipse Foundation. *Eclipse IDE.* Consulté le 28/09/2009.
http://www.eclipse.org.

[Eco] The Eclipse Foundation. *Eclipse Modeling Framework Project (EMF).* Consulté le 27/09/2009.
http://www.eclipse.org/modeling/emf/?project=emf.

[J2e] Sun Microsystems Inc. *Java EE.* Consulté le 27/09/2009.
http://java.sun.com/javaee/.

[Ker] IRISA – Projet Triskell. *Kermeta.* Consulté le 27/09/2009.
http://www.kermeta.org/.

[Ope] SGI and the Khronos Group. *OpenGL.* Consulté le 04/10/2009.
http://www.opengl.org/.

[Osg] Alliance OSGi. *Intergiciel à composants OSGi.* Consulté le 04/10/2009.
http://www.osgi.org.

[Swi] Sun Microsystems Inc. *Java Swing.* Consulté le 04/10/2009.
http://java.sun.com/docs/books/tutorial/uiswing/.

[Top] TOPCASED. *TOPCASED Modelling Tools.* Consulté le 28/09/2009.
http://www.topcased.org/.

Résumé

Cette thèse propose un pont entre les domaines des méthodes d'ingénierie des systèmes d'information à base de composants et des méthodes d'ingénierie des interfaces homme-machine (IHM) post-WIMP, dont font partie les interfaces de réalité mixte. Au fil du temps, ces domaines ont acquis une forte maturité, concrétisée par des démarches, modèles, outils et pratiques distinctes, autour de problématiques propres, respectivement les fonctionnalités du système et son utilisabilité. Il est par conséquent difficile de mettre en œuvre le développement de systèmes conjuguant pleinement ces deux aspects. Dans ce contexte, le transfert des systèmes interactifs en réalité mixte – intégrant des éléments virtuels dans le monde physique – des laboratoires vers les entreprises constitue une prise de risque importante.

Notre contribution est une extension de la méthode Symphony, issue du Génie Logiciel et ayant déjà fait ses preuves dans ce domaine, pour y intégrer les pratiques de l'IHM. Modèles et fragments de processus pour la collaboration entre des acteurs issus de cultures informatiques différentes sont intégrés à cette extension. Ainsi, au concept précédément défini d'Objet Métier, nous ajoutons celui d'Objet Interactionnel afin d'établir un espace commun de modélisation du système. Nous démontrons également l'impact que les choix d'interaction peuvent avoir sur l'organisation de l'espace métier et proposons des activités pour gérer l'évolution de ce dernier.

Nous décrivons enfin un ensemble d'outils pour documenter, transformer et exécuter la démarche et les produits de la méthode. Ceux-ci se basent sur les apports de l'ingénierie dirigée par les modèles (IDM) et la programmation orientée aspects pour construire des systèmes aux composants fortement découplés et réutilisables.

Mots-clefs : Méthode de développement, Réalité mixte, Processus, Modèle, Outil, Ingénierie dirigée par les modèles (IDM)

Abstract

This thesis proposes a bridge between the domains of component-based information systems engineering and post-WIMP human-computer interfaces engineering. In particular, our contributions focus on the design of mixed reality systems, which superimpose virtual elements on the physical world. These domains have progressively come to maturity, and have been refined into processes, models and distinct practices, which focus on specific problematics, respectively system functionalities and usability. Setting up a development process that fully integrates both of these aspects is therefore a challenging task. Furthermore, in this context, transferring the design of mixed reality interactive systems from the laboratories to the industry involves taking important risks.

Our contribution is an extension of the Symphony method, which was initially designed from an exclusively Software Engineering perspective. We integrated in this method the practices of the HCI domain, as well as models and process fragments for enabling collaborations between development actors with different computer science backgrounds. We added the concept of "Interactional Object" to the previously defined notion of "Business Object", in order to provide a common modelling ground. We also demonstrate the impact that interaction choices may have on the organization of the business space, and propose activities for managing its evolution.

Finally, we describe a set of tools for documenting, transforming and executing the process and/or method products. These tools capitalize on the assets from Model Driven Engineering (MDE) and Aspect-Oriented Programming (AOP) for building highly decoupled and reusable component-based systems.

Keywords : Development method, Mixed reality, Process, Model, Tool, Model Driven Engineering (MDE)