Une approche MDA pour l'intégration de la personnalisation du contenu dans la conception et la génération des applications interactives

Firas Bacha

Université de Valenciennes et du Hainaut-Cambrésis

Thèse de doctorat

Pour obtenir le grade de Docteur de l'Université de

VALENCIENNES ET DU HAINAUT-CAMBRESIS

Discipline, spécialité selon la liste des spécialités pour lesquelles l'Ecole Doctorale est accréditée :

Sciences et Technologie, Mention : Informatique

Présentée et soutenue par Firas, BACHA.

Le 16/05/2013, à Valenciennes

Ecole doctorale :
Sciences Pour l'Ingénieur (SPI)

Equipe de recherche, Laboratoire :
Décision, Interaction et Mobilité (DIM)
Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines
(LAMIH UMR CNRS 8201)

Une approche MDA pour l'intégration de la personnalisation du contenu dans la conception et la génération des applications interactives

JURY

Président du jury
- AIT AMEUR, Yamine. Professeur des Universités. Institut National Polytechnique de Toulouse.

Rapporteurs
- PALANQUE, Philippe. Professeur des Universités. Université Paul Sabatier, Toulouse III.
- ANIORTE, Philippe. Professeur des Universités. Université de Pau et des Pays de l'Adour.

Examinateurs
- AIT AMEUR, Yamine. Professeur des Universités. Institut National Polytechnique de Toulouse.
- BASTIEN, Christian. Professeur des Universités. Université de Lorraine.
- GARGOURI, Faïez. Professeur des Universités. Université de Sfax.

Directeur de thèse
- ABED, Mourad. Professeur des Universités. Université de Valenciennes.

Co-directrice de thèse
-MARCAL DE OLIVEIRA, Káthia. Maître de Conférences. Université de Valenciennes.

Membres invités
- LACRAMPE, Stéphane. Directeur de Obéo. Carquefou.

Une approche MDA pour l'intégration de la personnalisation du contenu dans la conception et la génération des applications interactives

Résumé

Les travaux de recherche présentés dans ce mémoire se situent dans les thématiques de la génération des applications interactives et de la personnalisation du contenu. Cette thèse propose une approche de type MDA (Model Driven Architecture), indépendante du domaine d'application, permettant la conception et la génération semi-automatique des applications interactives à contenus personnalisés, compte tenu des informations sur le contexte d'utilisation et l'ontologie de domaine. Cette approche met en œuvre deux méthodes de personnalisation du contenu, à savoir le remplissage automatique des formulaires et l'enrichissement des requêtes. Pour atteindre cet objectif, nous avons développé la solution technique permettant la conception, la transformation des modèles ainsi que la génération de l'IHM (Interface Homme-Machine) finale.

Mots-clés :

Model Driven Architecture – Personnalisation – Interface Homme-machine – Contexte – Ontologie

---

An MDA approach for content personalization integration in the design and the generation of interactive applications

Abstract

The research work presented in this thesis belongs to the fields of interactive applications generation and content personalization. This thesis proposes an MDA (Model Driven Architecture) approach, independent of the domain application, allowing the design and the semi-automatic generation of personalized content interactive applications. This generation relies on context information and the domain ontology. This approach implements two content personalization methods ; namely the forms auto-filling and the automatic queries enrichment. To achieve this goal, we developed the technical solution allowing the design, the models transformations as well as the generation of the final HCI (Human-Computer Interface).

Keywords :

Model Driven Architecture – Personalization – Human-Computer Interface – Context – Ontology

Avant propos

Cette thèse doit beaucoup aux nombreuses personnes qui m'ont encouragé, soutenu et conforté au long de toutes ces années. Qu'elles trouvent dans ce travail l'expression de mes plus sincères remerciements.

Je tiens à remercier Messieurs Philippe Palanque, Professeur à l'Université Paul Sabatier-Toulouse III et Philippe Aniorté, Professeur à l'Université de Pau et des Pays de l'Adour pour m'avoir fait l'honneur d'être rapporteurs de ce mémoire.

Je suis également très reconnaissant envers Messieurs Yamine Ait Ameur, Professeur à Institut National Polytechnique de Toulouse, Christian Bastien, Professeur à l'Université de Lorraine et Faïez Gargouri, Professeur à l'Université de Sfax, pour avoir accepté d'examiner ce travail.

Je souhaite renouveler mes remerciements à Mourad Abed pour avoir proposé et encadré cette thèse.

J'aimerai aussi remercier ma codirectrice de thèse, Káthia Marcal De Oliveira, pour sa disponibilité déconcertante (weekends et jours fériés, ou même la nuit) et pour son aide précieuse à la rédaction de mes articles ou de ma thèse. La justesse de ses critiques a été très constructive et utile. Je lui exprime ma très profonde gratitude !

La thèse a parfois été un moment difficile pour mes proches. Elle est très preneuse de temps ! Et j'avoue ne pas leurs avoir consacré le temps qu'ils le méritent. Myriam, je te remercie pour tes encouragements et ton soutien. Malgré la période difficile qu'on a traversé, tu m'as été toujours disponible et compréhensive... Un remerciement spécial pour mes parents sans qui je n'aurai jamais eu l'opportunité de continuer mes études en France et effectuer cette thèse. Je m'adresse également à mon frère Mlouki en le remerciant pour les plats si délicieux qu'il m'a préparés sur Paris ! J'ai également pu compter sur l'aide et l'encouragement de Tata Hakima et 3am Mokhtar.

Je remercie aussi mes amis, spécialement Abir et Salem, pour tous les bons moments passés ensemble durant la thèse. Merci pour m'avoir accompagné et aidé pendant ces dernières années.

MERCI à ceux qui ont toujours cru en moi et j'espère être toujours à leur hauteur.

À mon trésor,

À mes parents,

À mon frère,

À ma famille,

À mes amis.

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Motivations

De nos jours, la croissance continue de l'utilisation des IHM (Interfaces Homme-Machine) ainsi que la diversité de leurs modes d'interaction permettent aux utilisateurs d'accéder aux informations n'importe où et à tout moment. Cette flexibilité rend les utilisateurs plus exigeants et apporte de nouveaux défis à ces IHM. La pertinence des informations fournies et leur adaptation aux préférences des utilisateurs sont devenues des facteurs clés de succès ou de rejet des IHM ; il s'agit donc de conquérir les utilisateurs en leur fournissant des IHM personnalisées et adaptées à leurs besoins. De ce fait, pour qu'elles restent utilisables, malgré leur complexité croissante, les IHM doivent subir des mutations profondes afin de répondre aux nouvelles exigences. La littérature actuelle montre un intérêt croissant pour la création des IHM personnalisées, sensibles au contexte. Lorsque nous parlons de la sensibilité au contexte, nous entendons des applications capables de fournir à un utilisateur, à chaque instant de l'interaction, des contenus et des services adaptés à ses besoins et à ses attentes. Pour ceci l'adaptation utilise l'ensemble des informations concernant le contexte d'utilisation, représenté par le triplet $\prec$Utilisateur ; Environnement ; Plateforme$\succ$.

Par ailleurs, l'IDM (Ingénierie Dirigée par les Modèles) est aujourd'hui en passe de devenir le nouveau paradigme en matière de développement d'application. L'objectif derrière l'utilisation d'un tel paradigme est d'augmenter la productivité et réduire le temps du développement des systèmes complexes au moyen de modèles qui sont beaucoup moins liés à la technologie et qui sont beaucoup plus proches du domaine métier. Cette abstraction des problèmes complexes, rend les systèmes plus faciles à spécifier et à maintenir. L'une des variantes les plus connues de l'IDM est le standard MDA (Model Driven Architecture) qui, à travers l'utilisation de trois niveaux d'abstraction, a pour but de séparer la logique métier de l'application de la technologie qui sera utilisée pour la réaliser. L'objectif est de permettre de supprimer le lien direct entre les applications et le codage qui leur est associé, facilitant leur interopérabilité et les rendant ainsi moins sensibles aux évolutions technologiques.

Problématique

Dernièrement, avec l'amélioration des technologies utilisées et l'accélération du processus de conception, l'IDM a attiré l'attention de la communauté d'IHM (Interaction Homme-Machine) pour la conception et la génération des systèmes interactifs personnalisés. Dans ce cadre, les approches proposées s'intéressent généralement à la personnalisation de la présentation des éléments de l'interface (champs, résolution et taille de l'écran, etc.), appelée personnalisation du contenant, se basant sur quelques informations du contexte d'utilisation. Cependant, pour réellement parvenir à mettre en œuvre la personnalisation, il est important de considérer non seulement le contenant mais aussi le contenu, représentant l'ensemble des informations pertinentes fournies à l'utilisateur pendant son interaction

avec l'IHM, compte tenu de son contexte d'utilisation. La problématique à laquelle nous cherchons à apporter une solution est la suivante :

> ***Comment intégrer la personnalisation du contenu dans la conception et la génération des applications interactives en se basant sur une approche de type MDA ?***

Objectifs de cette thèse

Cette thèse répond à l'interrogation précédente en présentant une approche de type MDA, indépendante du domaine d'application, permettant la conception et la génération semi-automatique des applications interactives à contenus personnalisés, compte tenu des informations sur le contexte d'utilisation. Pour atteindre cet objectif, le modèle de contexte et l'ontologie de domaine ont été considérés comme éléments centraux de conception, de transformation des modèles et de génération de l'IHM finale. Cette approche met en œuvre deux méthodes de personnalisation, à savoir le remplissage automatique des formulaires et l'enrichissement des requêtes.

Organisation de la thèse

Le présent mémoire comporte deux parties principales : la première partie constitue une étude bibliographique et elle est composée des deux premiers chapitres. La deuxième partie, qui comporte les trois derniers chapitres, regroupe l'ensemble de nos contributions.

Dans le premier chapitre, nous explorons une introduction au domaine de l'ingénierie dirigée par les modèles, puis nous présentons deux notions clés de cette thèse : la personnalisation et le contexte. Étant donné que nous nous intéressons à la personnalisation du contenu, nous définissons également les ontologies, leurs classifications ainsi que les différents langages et outils permettant leur manipulation. Ensuite nous présentons un état de l'art sur les différents modèles des tâches. Pour décrire une interface, à n'importe quel niveau d'abstraction, il est nécessaire d'utiliser un langage de description des interfaces. De ce fait, nous terminons ce chapitre par une étude comparative entre les différents langages permettant de décrire une IHM.

Dans le deuxième chapitre, nous présentons un état de l'art sur les approches basées sur les modèles visant la génération des IHM sensibles au contexte. Afin de bien étudier ces travaux et pour positionner notre travail parmi eux, nous avons répertorié les critères sur lesquels se base cette étude en trois volets majeurs : les critères orientés modélisation qui permettent de décrire l'ensemble des modèles d'une approche bien déterminée ; les critères orientés personnalisation qui servent à évaluer la réussite de l'approche en terme de mise en œuvre de la personnalisation ; et finalement les critères orientés implémentation permettant de mesurer la maturité de l'approche, au niveau technique, en terme d'outillage et

d'implémentation des modèles proposés. Cette étude est suivie d'une synthèse permettant de classifier les propositions étudiées selon différentes perspectives.

Dans le troisième chapitre, nous présentons l'architecture MDA de notre approche dont le but est de prendre en compte la personnalisation du contenu dès les premières phases de conception. Le modèle de contexte et l'ontologie de domaine constituent les éléments clés de cette approche. De ce fait, nous commençons par décrire notre approche d'une manière globale. Ensuite, nous détaillons les différents modèles qui la construisent et la manière avec laquelle ils sont pris en compte pour concevoir et générer des IHM à contenus personnalisés. Finalement, nous présentons les différentes étapes à suivre pour passer d'un niveau MDA à un autre, jusqu'à la génération de l'IHM finale.

Afin de valider cette approche et de montrer son intérêt fonctionnel, nous sommes tenus de fournir l'ensemble des outils permettant de la mettre en pratique. Pour chaque étape de notre approche, nous avons choisi les technologies et les outils existants, qui conviennent le mieux à nos besoins. Le quatrième chapitre aborde l'ensemble des développements réalisés dans le cadre de cette thèse, permettant de créer, de manipuler et de transformer l'ensemble des modèles conceptuels de notre approche. Nous avons eu recours principalement à trois outils qui forment le cœur d'instrumentation de notre proposition : EMF (Eclipse Modeling Framework), ATL (Atlas transformation Language) et LiquidApps. Nous commençons par présenter l'architecture globale de l'atelier logiciel composé des outils précédents ainsi que les décisions méthodologiques prises et les techniques choisies. Nous détaillons ensuite le cadre de métamodélisation de cette proposition, la logique de transformation des modèles et l'ensemble des règles développées pour mettre en œuvre l'ensemble de ces transformations.

Le cinquième chapitre concerne la mise en pratique de notre approche, en proposant deux études de cas permettant de générer des IHM personnalisées. La première application traite un système d'information pour les usagers du transport en commun et la deuxième représente un système d'assistance médicale. Ce choix de deux domaines d'application différents illustre la généricité de notre approche et son indépendance de tout domaine d'application.

Ce mémoire se termine par une conclusion portant à la fois sur le bilan de notre recherche, sur l'ensemble des contributions apportées par cette thèse et finalement ses limites qui représentent les perspectives tracées pour la suite de ce travail.

Première partie

Étude bibliographique

Chapitre 1

Conception et personnalisation des IHM basées sur les modèles

Introduction

L'IHM (Interaction Homme-Machine) est la discipline consacrée à la conception, la mise en œuvre et l'évaluation des systèmes informatiques interactifs destinés à des utilisateurs ainsi qu'a l'étude des phénomènes majeurs qui les entourent (Hewett *et al.*, 1992). Afin de comprendre parfaitement l'utilisation d'une technologie, nous devons étudier l'être humain, la technologie et l'interaction entre les deux. Pour cela l'IHM est un champ multidisciplinaire, qui regroupe des experts de différents domaines tels que l'ergonomie, l'automatique, l'informatique ainsi que la psychologie. Cependant, cette pluridisciplinarité accroit le processus du développement et le rend plus complexe compte tenu des contraintes et des exigences de chaque domaine. En plus, avec les avancées technologiques que connaissent les systèmes interactifs en association avec les nouvelles exigences des utilisateurs, l'adaptation des interfaces utilisateur est un aspect primordial à prendre en compte pendant la conception des interfaces utilisateur.

Une des solutions pour réduire cette complexité consiste ainsi à mettre en relation les spécificités de ces différentes disciplines pour aboutir à des méthodes qui couvrent les différents aspects de l'interface. (Javier, 2010) a classifié les approches du développement des systèmes interactifs en 3 groupes :

- **L'approche par programmation** : Elle produit la représentation de l'interface utilisateur par le biais d'un langage de programmation procédural, orienté objet ou déclaratif.
- **L'approche exploratrice** : Elle est basée sur le développement des maquettes des interfaces et la possibilité d'en générer automatiquement le code source. Cette approche tire parti des environnements de développement où il est possible de construire facilement une maquette de l'interface.
- **L'approche basée sur les modèles** : Cette méthode propose de spécifier une application d'une manière abstraite à travers un ensemble de modèles conceptuels. A partir de ces spécifications, en leur appliquant une suite de transformations automatiques ou semi-automatiques, l'interface finale sera générée. L'automatisation de certaines parties du processus de développement réduit l'intervention du concepteur et assure une qualité fiable des interfaces obtenues. Ce type d'approche sera détaillé dans le chapitre 1, §1.1.

Rappelons que dans le cadre de notre thèse, nous cherchons à générer des applications interactives à contenus personnalisés, en suivant une approche de type MDA (Model-Driven Architecture). De ce fait, nous proposons à travers ce chapitre d'explorer les éléments de base constituant cette thèse. En effet, dans un premier temps, nous introduisons le domaine de l'ingénierie dirigée par les modèles, discipline sur laquelle repose MDA. Ensuite, étant donné que nous nous intéressons à la personnalisation du contenu, nous consacrons une section pour introduire la notion de personnalisation ainsi que la notion de contexte, un facteur clé pour la mise en œuvre de la personnalisation. Les ontologies permettent de représenter formellement les connaissances et de décrire leurs raisonnements. Compte tenu de notre cible de personnalisation, à savoir le contenu, la quatrième section sera consacrée à la définition des ontologies, leurs classifications ainsi que les différents langages et outils permettant leurs manipulations. La cinquième section, présente un état de l'art sur

les différents formalismes de la modélisation des tâches, constituant les points d'entrée communs pour la conception des applications interactives basées sur les modèles. Nous terminons ce chapitre par une étude comparative entre les différents langages permettant de décrire une IHM. En effet, pour décrire une IHM, à n'importe quel niveau d'abstraction, il est nécessaire d'utiliser un langage de description des interfaces (Florins *et al.*, 2006).

1.1 L'ingénierie dirigée par les modèles

L'IDM (Ingénierie Dirigée par les Modèles) se réfère à l'utilisation systématique des modèles comme des éléments centraux tout au long du cycle de vie du logiciel. L'objectif derrière l'utilisation d'un tel paradigme est d'*augmenter la productivité* et *réduire le temps de développement des systèmes complexes* au moyen de modèles qui sont beaucoup moins liés à la technologie et qui sont beaucoup plus proches du domaine. Cette abstraction des problèmes complexes rend les systèmes plus faciles à spécifier et à maintenir. Cette section sera consacrée à l'introduction de cette approche et les notions de base sur lesquelles elle repose.

1.1.1 Définitions

Même si l'IDM s'appuie sur les modèles qui sont considérés comme les piliers de cette approche, il n'y a pas de consensus concernant la définition de la notion du modèle. (Minsky, 1969) considère le modèle comme étant "*...une représentation (ou abstraction) d'un système, décrit dans une intention particulière*". (Bézivin et Gerbé, 2001) définissent un modèle comme "*... une simplification d'un système construit avec un objectif bien déterminé. Le modèle devrait être capable de répondre aux questions à la place du système actuel*". Dans (Seidewitz, 2003), un modèle est considéré comme "*... un ensemble de définitions concernant un système qui est en train d'être étudié*". Selon (Mellor *et al.*, 2003) c'est "*... un ensemble cohérent des éléments formels décrivant quelque chose (par exemple un système, une banque, un téléphone, ou un train), construit dans un but bien déterminé*". De notre part, nous avons retenu la définition proposée par (OMG, 2003) :

Définition. *Un modèle est une description ou une spécification d'un système et de son environnement dans un but bien déterminé. Un modèle est souvent présenté comme une combinaison de dessins et de textes. Le texte peut être dans un langage de modélisation ou dans une langue naturelle*

Dans le domaine de l'IDM, la méta-modélisation joue un rôle très important. En effet, elle est considérée comme une technique courante pour définir la syntaxe abstraite des Modèles et des interrelations entre les éléments du modèle. Si le modèle est une abstraction des éléments du monde réel, le méta-modèle représente encore une autre abstraction, définissant les propriétés du modèle lui-même. Un modèle est dit conforme à son métamodèle. (OMG, 2006) a défini le métamodèle comme suit :

Définition. *Un métamodèle est un modèle définissant le langage permettant d'exprimer un modèle.*

À cet égard, l'OMG [1] (Object Management Group) a introduit l'architecture à quatre niveaux illustrée dans la Figure 1.1 appelée également *l'architecture 3+1* ou bien *la pile de modélisation*. Au niveau inférieur, la couche M0 représente le système réel. Un modèle représente ce système au niveau M1. Ce modèle est conforme à son méta-modèle défini au niveau M2 et le méta-modèle lui-même est conforme à son méta-métamodèle au niveau M3. La définition du méta-métamodèle est réflexive étant donné qu'il est conforme à lui même. L'OMG a proposé Meta-Object Facility (MOF) (OMG, 2006) comme un standard pour la spécification des méta-métamodèles et a défini le méta-métamodèle comme suit :

Définition. *Un méta-métamodèle est un modèle qui permet de décrire un langage de métamodélisation. Un méta-métamodèle doit être réflexif pour limiter le nombre de niveaux d'abstraction.*

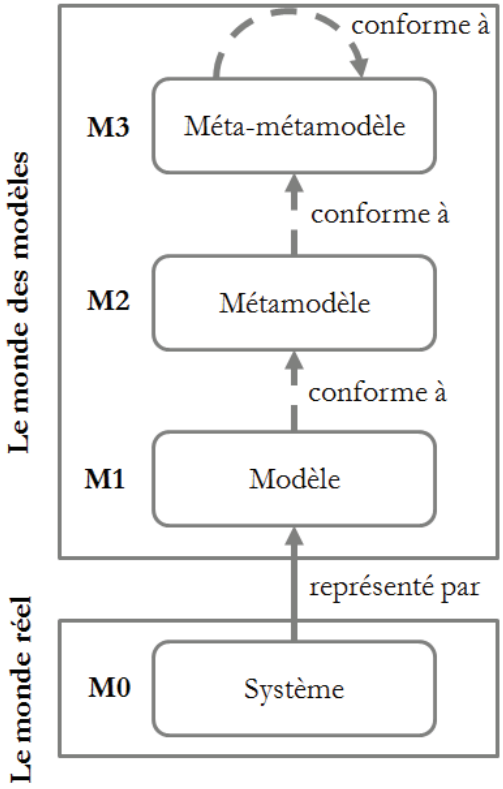


FIGURE 1.1 – Architecture 3+1 adaptée de (Bézivin, 2005)

1. http ://www.omg.org/

1.1.2 Model-Driven Architecture

MDA (Model-Driven Architecture) est un standard, lancé par l'OMG (OMG, 2003), qui se base sur l'IDM, fournissant un ensemble de lignes directrices ainsi qu'une architecture pour la conception des systèmes logiciels.

L'approche MDA donne la possibilité de comprendre les systèmes complexes et le monde réel à travers une abstraction de ceux ci. Ce point de vue abstrait du système est élaboré dans un cadre conceptuel ainsi qu'un nombre de standards fournis par l'OMG permettant de définir les Modèles, leurs relations ainsi que leurs transformations (par exemple : UML (Unified Modeling Language), MOF et XMI (XML Metadata Interchange)). Pour représenter visuellement l'approche MDA, l'OMG a mis en place un *framework*, structuré de plusieurs types de Modèles. La Figure 1.2 représente le cycle de développement en Y, qui met en œuvre ces modèles ainsi que leurs relations :

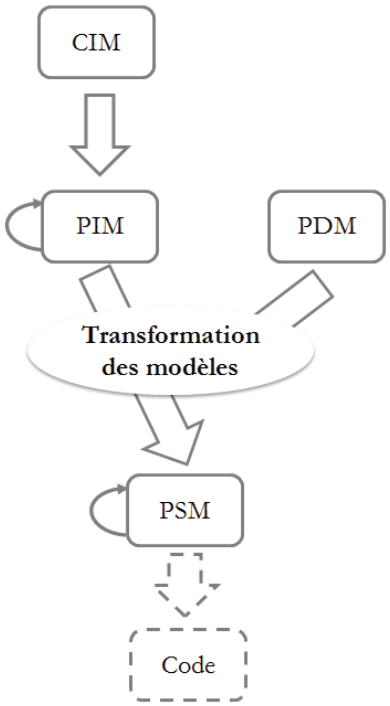


FIGURE 1.2 – Le cycle de développement en Y

– **CIM (Computation Independent Model)** : Ces modèles décrivent le système à concevoir d'un point de vue indépendant de l'informatisation. Le CIM permet une vision du système et de son environnement, tout en cachant les détails de structure et d'implémentation. Les Modèles du niveau CIM permettent de réduire l'écart entre les experts du domaine et entre les concepteurs. De ce fait, un modèle du CIM est parfois appelé un modèle de domaine.

– **PIM (Platform Independent Model)** : Les modèles du niveau PIM représentent une vision d'analyse et de conception du système, indépendamment de tout détail technologique concernant la plateforme (système d'exploitation, langage de programmation, matériel, performances du réseau, etc).
– **PSM (Platform Specific Model)** : Le niveau PSM présente une projection des modèles du niveau PIM vers une plateforme spécifique. Ces Modèles combinent les spécifications du PIM avec les détails propres à la plateforme.
– **PDM (Platform Description Model)** : Ces modèles décrivent la plateforme sur laquelle le système va être exécuté, en fournissant un ensemble de données techniques concernant les fonctionnalités et l'utilisation de la plateforme.

1.1.3 Transformation des modèles

Les modèles sont spécifiés à différents niveaux d'abstraction et parfois aussi en utilisant différentes spécifications. La transformation d'un ou plusieurs modèles sources vers un ou plusieurs modèles cibles, appelée la *transformation de modèles* représente un des piliers fondamentaux de l'IDM. (Kleppe *et al.*, 2003) définit la transformation de la manière suivante :

Définition. *Une transformation est une génération automatique d'un ou plusieurs modèles cibles à partir d'un ou plusieurs modèles sources, en respectant une définition de transformation. Une définition de transformation est un ensemble de règles de transformation qui décrivent la manière avec laquelle un modèle dans le langage source peut être transformé en un modèle dans le langage cible. Une règle de transformation est une description de la façon avec laquelle une ou plusieurs constructions dans le langage source peuvent être transformées en une ou plusieurs constructions dans le langage cible.*

Pour mettre en œuvre ce processus de transformation, un moteur de transformation prend en entrée un ou plusieurs modèle(s) conforme(s) à un (des) métamodèle(s) source(s) et produit en sortie un ou plusieurs autre(s) modèle(s) conforme(s) à un (des) métamodèle(s) cible(s). Le moteur de transformation, composé d'un ensemble de règles, doit être lui-même considéré comme étant un modèle. En conséquence, il est basé sur un métamodèle correspondant, qui est une définition abstraite du langage de transformation utilisé (cf. Figure 1.3).

(Mens et Gorp, 2006) proposent une taxonomie des transformations de modèles où ils définissent deux dimensions orthogonales : une *transformation horizontale* versus une *transformation verticale* et une *transformation endogène* versus une *transformation exogène* :
– Les transformations verticales des modèles sont utilisées pour affiner ou abstraire un modèle et, dans ce cas, les modèles sont situés dans des niveaux d'abstraction différents. Les transformations horizontales n'affectent pas l'abstraction des modèles et ils servent principalement à les restructurer étant donné que ces modèles appartiennent au même niveau d'abstraction ;

– Les transformations endogènes sont des transformations entre des modèles qui sont exprimés dans le même langage tandis que les transformations exogènes sont des transformations entre des modèles définis à l'aide des langages différents. (Mens et Gorp, 2006) qualifient les transformations endogènes par le terme *reformulation* et les transformations exogènes par le terme *traduction*.

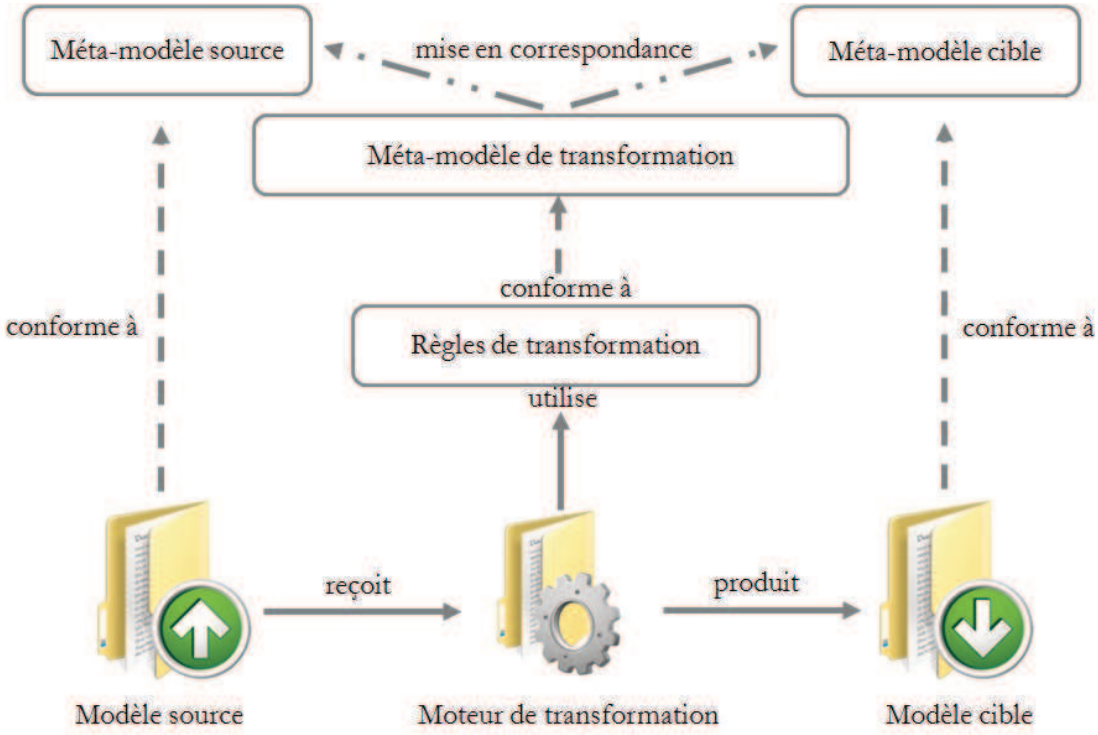


FIGURE 1.3 – Principe de la transformation des modèles en IDM (Touzi, 2007)

Il existe de nombreux langages et outils qui ont été proposés pour définir et exécuter les transformations. Selon (Czarnecki et Helsen, 2006; Diaw *et al.*, 2010), les langages de transformation peuvent être divisés en plusieurs catégories :

- Langages de programmation classiques : ex. JAVA
- Langage dédié d'un atelier de génie logiciel : ex. J de Objecteering [2]
- Langage lié à un domaine/espace technologique : ex. XSLT pour XML
- Langage/outil dédié à la transformation de modèles : ex. QVT et ATL
- Atelier de méta-modélisation avec langage d'action : ex. Kermeta

2. http ://www.objecteering.com/

Dans le cas des langages dédiés à la transformation des modèles, un langage peut être déclaratif, impératif ou hybride (Touzi, 2007) :
– Dans le cas d'un langage déclaratif, nous décrivons les métamodèles à traiter ainsi que les contraintes sur ces métamodèles. Les transformations s'exécutent sur une instance de ces métamodèles (un modèle), tout en respectant ces contraintes.
– Un langage impératif décrit comment le résultat est obtenu en imposant une suite d'actions élémentaires et détaillées que la machine doit effectuer.
– Un langage hybride est à la fois déclaratif et impératif.
Pour une comparaison entre les différents langages de transformation le lecteur peut se référer aux travaux de (Czarnecki et Helsen, 2003; Jouault et Kurtev, 2006).

1.2 La personnalisation des applications interactives

Ces dernières années, avec les avancées technologiques qu'ils connaissent, les systèmes informatiques soutiennent toute activité quotidienne en tenant en compte la mobilité de l'utilisateur. Cette flexibilité permet à l'utilisateur d'être plus exigeant. La solution est de conquérir les utilisateurs en développant des systèmes personnalisés qui s'adaptent à leurs besoins et à leurs contextes. Dans la suite nous définissons la notion de personnalisation et nous introduisons les différentes dimensions dans les-quelles elle s'applique ainsi que les différentes méthodes suivies pour la mettre en œuvre.

1.2.1 Définitions

Il n'existe pas de définition consensuelle de la notion de personnalisation. Généralement, les auteurs la définissent en fonction de leurs objectifs spécifiques et en fonction des applications (Doucet *et al.*, 2004; Anli, 2006). Cette notion est apparue dans le domaine de la Recherche d'Information (RI) et puis elle a été adoptée par quelques travaux dans le domaine de l'IHM. Pour (Hagen, 1999) la personnalisation est la capacité à fournir des contenus et des services qui sont adaptés aux personnes en fonction de leurs préférences et de leurs comportements. (Dyche, 2001) partage la définition précédente en indiquant qu'il faut également considérer l'adaptation de la communication lors de l'interaction. (Kim, 2002) ne définit pas la personnalisation par rapport à une seule personne et affirme qu'il s'agit de la livraison de l'information pertinente pour un groupe d'individus. Pour atteindre cette pertinence, cette information doit être récupérée puis transformée avant d'être livrée aux utilisateurs. En proposant une définition plus générique, (García-barrios *et al.*, 2005) introduisent la personnalisation comme étant l'adaptation par rapport à un utilisateur particulier. Ce dernier est caractérisé par un modèle qui permet de le décrire. (Simonin et Carbonell, 2007) définissent la personnalisation, dans le cadre des IHM, comme étant l'adaptation dynamique de l'interface en fonction du profil de l'utilisateur. Malgré la diversité des définitions, la personnalisation est souvent confondue avec les notions d'*adaptation* et de *customisation* (Mobasher *et al.*, 2000; Rossi *et al.*, 2001).

Personnalisation versus customisation

La personnalisation et la customisation visent à répondre de façon adaptée aux besoins et caractéristiques particuliers de chaque utilisateur (Cingil *et al.*, 2000). Étant un terme issu du domaine du commerce électronique, la customisation est une notion qui est moins utilisée dans le cadre de l'IHM. Selon (Rosenberg, 2001), la customisation est un processus contrôlé par l'utilisateur qui doit effectuer un choix parmi d'autres et c'est en fonction de ce choix que va réagir le système. En suivant une vision différente, dans le cas de la personnalisation, c'est le système qui construit sa propre connaissance de l'utilisateur (ses besoins, ses préférences, etc.) et essaie de lui fournir un contenu adapté à son profil.

Personnalisation versus adaptation

(Ledoux, 2001) définit l'adaptation comme étant le processus de modification des systèmes permettant de fonctionner d'une manière adéquate dans un contexte donné. Certains auteurs, (Anli, 2006; Kim, 2002), considèrent que la personnalisation fait partie de l'adaptation tandis que d'autres (Kappel *et al.*, 2000; Mobasher *et al.*, 2000; Ledoux, 2001) affirment que la personnalisation et l'adaptation sont synonymes. (Jameson, 2001) indique que l'adaptation inclue les notions d'*adaptabilité* et d'*adaptativité* et que la personnalisation peut être résumée par ces deux termes.

Les sens attribués à l'adaptabilité et l'adaptativité diffèrent selon les auteurs. Pour (Stephanidis *et al.*, 1998), l'adaptabilité fait référence à un processus d'adaptation basé sur les connaissances qu'a le système sur l'utilisateur avant le commencement de l'interaction. Dans ce cas, lors de son initialisation, le système fournira une version adaptée à l'utilisateur, qui restera inchangée au cours de la session d'utilisation. Dans la cas de l'adaptativité, les connaissances que possède le système sur l'utilisateur peuvent évoluer au cours des interactions. De ce fait, le système peut varier ses adaptations au cours de l'interaction avec l'utilisateur. L'adaptabilité peut être qualifiée d'*adaptation statique*, et inversement nous pouvons qualifier l'adaptativité, d'*adaptation dynamique* (Frasincar et Houben, 2002).

Selon d'autres auteurs (Dieterich *et al.*, 1993; Kobsa *et al.*, 2001), l'opposition des termes est basée sur le degré de contrôle que possède l'utilisateur pendant le processus d'adaptation. En effet, si le système permet à l'utilisateur le contrôle total du processus de l'adaptation, nous parlons de l'adaptabilité. Si le système effectue l'adaptation automatiquement sans intervention de l'utilisateur, alors nous sommes dans le cas de d'adaptativité. La Figure 1.4 illustre cette différence entre l'adaptabilité et l'adaptativité.

Dans ce mémoire, nous adoptons la définition de la personnalisation proposée par (Brossard, 2008) et (Bacha *et al.*, 2011e) :

Définition. *La personnalisation est la capacité de fournir à un utilisateur, à chaque instant, des contenus et des services adaptés à ses besoins et à ses attentes en utilisant des interactions homme-machine appropriées. Pour ce faire, cette adaptation utilise sa connaissance de l'utilisateur (caractéristiques et préférences), sa connaissance de l'environnement ainsi que des informations sur le comportement de l'utilisateur au moment de l'interaction.*

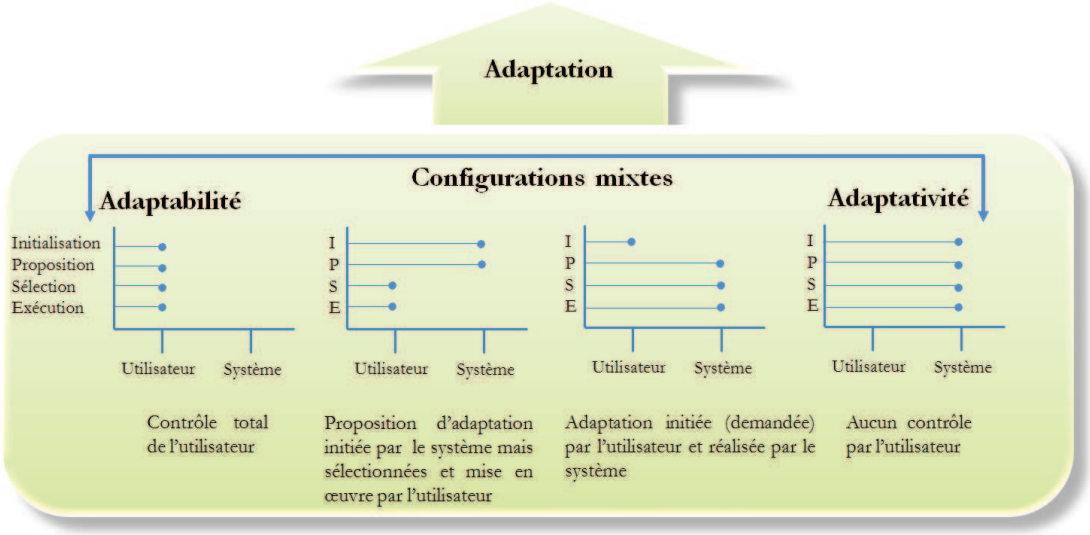


FIGURE 1.4 – Différents types de processus d'adaptation : de l'adaptabilité à l'adaptativité (Dieterich *et al.*, 1993)

1.2.2 Les dimensions de la personnalisation

La personnalisation peut prendre en compte plusieurs aspects et peut être appliquée à plusieurs niveaux (Kobsa *et al.*, 2001; Abbas, 2008) :

- *La présentation* : Cette catégorie de personnalisation a pour objectif d'adapter le style et le format des éléments d'interaction composant l'interface (les boutons, les champs de texte, etc.) compte tenu des besoins des utilisateurs et de leurs contextes. (Anli, 2006; Brossard, 2008) qualifient ce type de personnalisation de la personnalisation du contenant et affirment que dans ce cas nous parlons de la plasticité de l'interface telle qu'elle a été définie par (Thevenin et Coutaz, 1999). Ainsi, la plasticité est *"la capacité des interfaces de s'adapter à leur contexte d'usage dans le respect de leur utilisabilité. Le contexte d'usage se définit comme le triplet utilisateur, plateforme et environnement"*.
- *La structure* : Ce type de personnalisation s'applique sur les liens reliant les pages d'un site web. Par exemple, certains liens peuvent être proposés avec une annotation

particulière ou mis au premier rang dans une liste de liens selon leur pertinence.

- *Les fonctionnalités* : Elles consistent à identifier les besoins fonctionnels que cherche l'utilisateur et faciliter l'accomplissement d'une tâche en adaptant le système. Par exemple, le système peut éliminer certaine fonctionnalités si l'utilisateur n'en a pas besoin.
- *La navigation* : Ce type de personnalisation a pour but d'adapter le système en fonction de sa connaissance de l'utilisateur, permettant de l'orienter, pour éviter les chemins menant à des informations non pertinentes.
- *Le contenu* : Il s'agit de sélectionner et adapter les informations pertinentes à l'utilisateur, ses préférences ainsi que son contexte. (Brossard, 2008) affirme que dans ce cas nous parlons de context-awareness (cf. chapitre 1, §1.3.3).

La Figure 1.5 illustre des exemples de personnalisation du contenu et du contenant d'un système d'information pour les voyageurs.

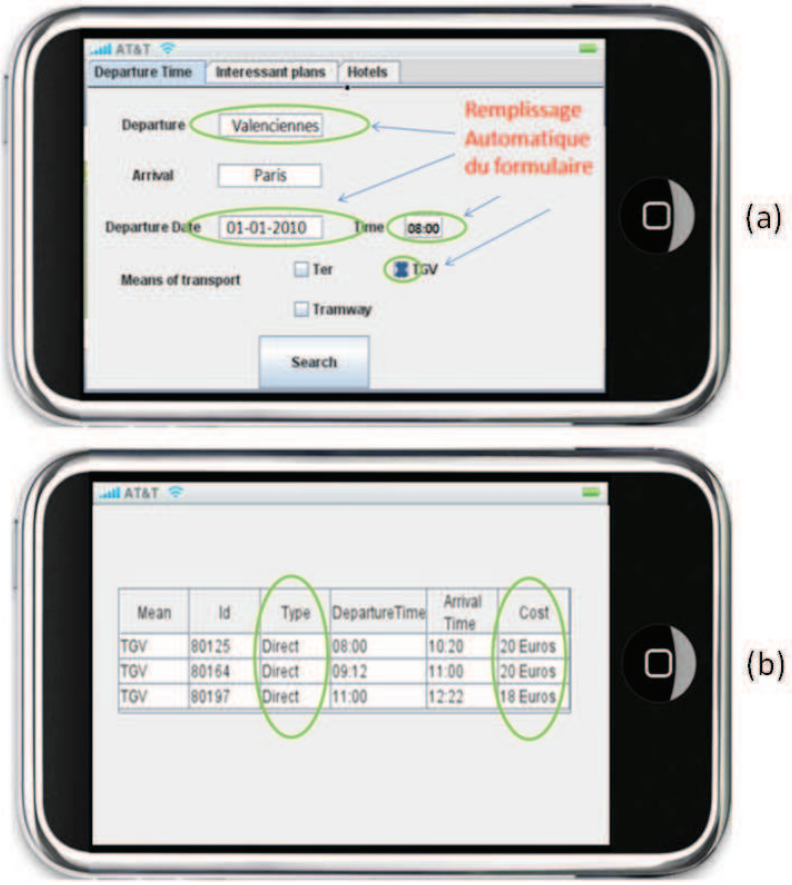


FIGURE 1.5 – Exemples de la personnalisation d'une interface utilisateur dans un système d'information pour le transport en commun

Par exemple, l'adaptation de la taille des éléments de l'interface tels que les polices et les widgets représente une personnalisation du contenant en fonction de la plateforme utilisée par l'utilisateur (dans ce cas, un IPhone(R)). Les informations concernant la ville et la date de départ (cf. Figure 1.5(a)) sont des exemples de personnalisation du contenu. En effet, la ville de départ est automatiquement complétée en fonction de l'emplacement de l'utilisateur au moment de l'interaction avec l'interface. La date de départ est remplie en fonction de la date du jour actuel. Un autre exemple de la personnalisation du contenu est présenté dans la Figure 1.5(b). Dans cet exemple, supposons que le système ait connaissance que l'utilisateur est âgé de moins de 18 ans et incapable de marcher, il lui proposera uniquement des itinéraires directs avec des prix réduits (selon son âge).

Dans le cadre de nos travaux, qui ont pour objectif d'intégrer la personnalisation du contenu dans la conception des applications interactives en se basant sur une approche MDA et en tenant compte des informations sur le contexte, nous ne nous intéresserons dans la suite qu'a la personnalisation des contenus.

1.2.3 Les facteurs influençant la personnalisation du contenu

Même si la personnalisation peut se faire au niveau de plusieurs dimensions, chacune d'elles dépend d'un ensemble de facteurs qui influe le processus de personnalisation. (Van Setten, 2001) a identifié quatre groupes de facteurs dont dépend la personnalisation du contenu (cf. Figure 1.6) :

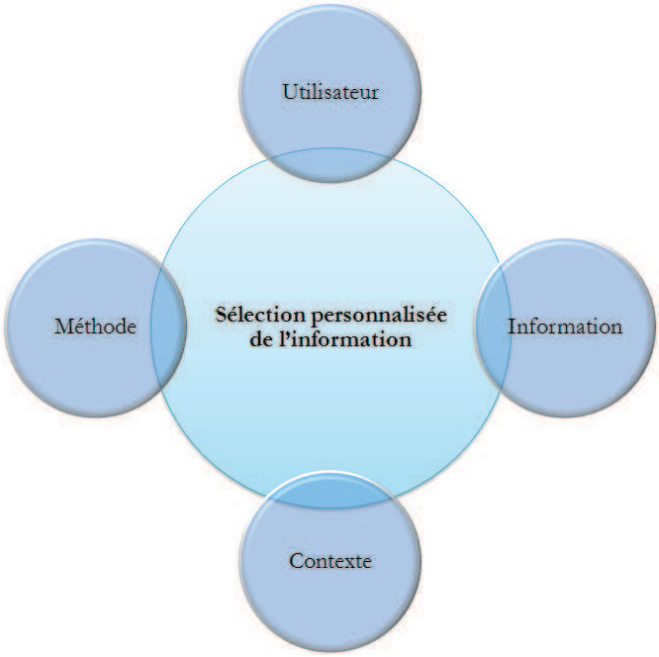


FIGURE 1.6 – Les facteurs influençant la personnalisation du contenu (Van Setten, 2001)

- *Les facteurs liés aux utilisateurs* : Ils déterminent la maniéré avec laquelle l'information doit être personnalisée compte tenu des centres d'intérêt des utilisateurs, leurs caractéristiques, leurs préférences, etc.
- *Les facteurs d'information ou du contenu* : Ils représentent l'ensemble des propriétés caractérisant les contenus et permettant de décrire la manière avec laquelle l'information peut être personnalisée. Le type d'un document et son méta-data associé sont des exemples de propriétés du contenu.
- *Les facteurs de contexte* : Ce sont tous les facteurs qui s'intéressent à décrire l'environnement technique et physique du système et de l'utilisateur. Par exemple, la localisation de l'utilisateur, la plateforme d'interaction, etc.
- *Les facteurs liés à la méthode de personnalisation* : Le résultat d'une opération de personnalisation dépend fortement de la méthode utilisée. Dans la suite nous présenterons quelques méthodes utilisées pour la personnalisation du contenu.

1.2.4 Les méthodes et les services de la personnalisation

Les différentes dimensions de personnalisation présentées précédemment peuvent être réparties sur trois domaines d'application :
– L'IHM (Interaction Homme-Machine)
– La RI (Recherche d'Information)
– Les BD (Bases de données)
Pour la mise en place de la personnalisation du contenu, plusieurs méthodes et services existent. D'après (Doucet *et al.*, 2004; Ioannidis et Koutrika, 2005; Abbas, 2008), parmi celles les plus couramment utilisées, nous pouvons citer :

- *Le filtrage de l'information* : Le principe est d'éliminer les données non pertinentes sur un ensemble d'informations sollicitées par l'utilisateur.
- *La recommandation* : Elle consiste à proposer et à recommander à l'utilisateur un ensemble d'informations sous différentes formes (vidéo, images, liens, etc.) en se basant sur les expériences des autres utilisateurs avec le système (Burke, 2002; Miller *et al.*, 2004).
- *La recherche personnalisée de l'information* : elle peut être réalisée en suivant deux méthodes :
  – L'enrichissement des requêtes qui permet d'agir sur la requête pour effectuer la personnalisation en tenant compte du profil de l'utilisateur. Cette opération d'enrichissement s'effectue de deux manières :
    – En enrichissant le langage de la requête par des termes à travers lesquels nous pouvons exprimer les préférences de l'utilisateur. Cette idée a été créée par (Lacroix et Lavency, 1987) et ensuite plusieurs travaux sont apparus dans la même lignée (Börzsönyi *et al.*, 2001; Chomicki, 2002; Kießling, 2002).
    – En augmentant la requête initiale en lui ajoutant de nouvelles conditions sans augmenter le langage lui-même, et ce, à partir des informations issues de profil

d'utilisateur (Koutrika et Ioannidis, 2005), ou en l'enrichissant avec des synonymes et des mots reliés sémantiquement aux termes de la requête à partir d'un thésaurus ou une ontologie (Manning *et al.*, 2008).

– Le classement des résultats (Result Ranking) qui permet de faire le tri des informations pertinentes, depuis une liste des résultats retournes à l'utilisateur, en tenant compte de son profil.

- *Le remplissage automatique des formulaires* : Actuellement les formulaires de tous les types sont utilisés dans la plupart des systèmes d'information, et la plupart d'entre eux sont remplis manuellement par les utilisateurs. Parmi les services de personnalisation qu'on peut déployer notamment en Web, nous citons le service du remplissage automatique des formulaires tenant compte du profil de l'utilisateur et son contexte. L'importance de ce service a été mise en évidence par plusieurs auteurs tel que (Flaten, 2007; Rukzio *et al.*, 2008; Tolulope, 2008).

Comme mentionné précédemment, la personnalisation du contenu est intimement liée à l'utilisateur et au contexte dans lequel se déroule l'interaction. La section suivante sera consacrée pour introduire cette notion et pour souligner son influence sur la personnalisation des IHM.

1.3 Le contexte en Interaction Homme-Machine

1.3.1 Définitions

Le contexte est un concept qui apparait dans plusieurs domaines (Bradley et Dunlop, 2005). En informatique, la notion de contexte est apparue pour la première fois dans le domaine de l'IA (Intelligence Artificielle) (Mccarthy, 1993) où elle a été utilisée pour diviser une base de connaissances en un ensemble de constructions logiques qui facilitent le raisonnement. Plus récemment, avec les progrès de l'informatique mobile, le contexte est devenu un sujet d'intérêt pour d'autres domaines de la science informatique, et particulièrement pour l'Interaction Homme-Machine.

Dans le domaine de l'IHM, donner une définition au contexte a fait l'objet de plusieurs tentatives. (Zimmermann *et al.*, 2007) affirment que le contexte est une notion difficile à définir et que la majorité des définitions existantes peuvent être classées en définitions par les synonymes et définitions par les exemples.

En IHM, l'apparition de la notion du contexte datait depuis l'année 1994 avec (Schilit *et al.*, 1994) qui ont introduit la notion du *sensibilité au contexte* (*context-awareness* en anglais). Les auteurs considèrent qu'un système est sensible au contexte s'il utilise le contexte pour fournir à l'utilisateur des informations ou des services pertinents. Dans ce cas, le contexte est capable de répondre aux questions suivantes : où vous êtes ? Avec qui vous êtes ? Et quelles sont les ressources proches de vous ?

(Brown, 1996) s'est contenté des objets de l'environnement pour définir le contexte. Tandisque, (Brown *et al.*, 1997) ont introduit l'identité de l'utilisateur et sa localisation ainsi

que les conditions météorologiques, la température et la saison. La même perception du contexte a été partagée avec (Ryan *et al.*, 1997).

En 1997, (Ward *et al.*, 1997) ont conservé globalement les mêmes éléments du contexte définis par (Schilit *et al.*, 1994), en s'intéressant aussi à l'emplacement des autres objets entourant l'utilisateur.

(Pascoe, 1998) a défini le contexte comme étant le sous-ensemble d'états physiques et conceptuels d'une entité particulière. L'auteur a fait réapparaitre la notion du context-awareness en décrivant la sensibilité au contexte comme étant la capacité des dispositifs informatiques à détecter, interpréter et répondre aux aspects de l'environnement local d'un utilisateur et les appareils informatiques eux mêmes.

En 1999, la notion du contexte commence à se généraliser avec la définition proposée par (Dey *et al.*, 1999) qui considèrent le contexte comme étant toute information permettant de caractériser une situation ou une entité sachant qu'une entité peut être une personne un endroit ou un objet pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application. Dans cette définition, les auteurs ont veillé à ne pas se focaliser sur un élément spécifique pour définir le contexte, mais plutôt à prendre en compte tous les éléments implicites et explicites pouvant influencer l'interaction.

Dans le but de détailler la définition précédente, (Schmidt *et al.*, 1999) affirment que le contexte permet de décrire une situation ainsi que l'environnement dans lequel se trouve l'utilisateur et sa plateforme d'interaction. Les auteurs ont créé un espace de caractéristiques hiérarchiquement organisé qui contient les éléments qui peuvent définir le contexte. Au sommet de cette hiérarchie, nous trouvons deux facteurs principaux : les facteurs humains et l'environnement physique.

(Thevenin et Coutaz, 1999), avec leur proposition de la notion de la plasticité, qualifient le contexte par le terme contexte d'interaction. Cette proposition a été effectuée dans le cadre de la prise en compte du contexte dans les approches se basant sur l'IDM. Le contexte d'interaction tel qu'il a été proposé par les auteurs est décrit à travers le triplet $\prec$Objet ; Utilisateur ; Environnement$\succ$.

En 2000, (Lieberman et Selker, 2000) ont proposé une nouvelle définition du contexte en le considérant comme étant tout ce qui affecte l'informatisation, sauf les entrées et les sorties explicites. Opposée à la définition proposée par (Schilit *et al.*, 1994), cette définition est très technique et elle est centrée sur l'application au lieu de l'utilisateur. L'entrée explicite est l'entrée qui vient directement de l'utilisateur sous forme de commandes passées via l'interface utilisateur. Le contexte est donc toute information que le système détecte au delà des commandes directes et qui a un effet sur l'état du système.

Bien que (Winograd, 2001) considère que la définition proposée par (Dey *et al.*, 1999) est la plus complète de l'ensemble des définitions disponibles, il critique le fait qu'elle soit trop générique et considère qu'elle n'est pas limitée et manque de précision. Pour apporter plus de spécificité, (Winograd, 2001) redéfinit le contexte en le considérant comme étant un ensemble d'informations, partagées et structurées.

Avec (Calvary *et al.*, 2001) le contexte est défini sous le nom du contexte d'utilisation. Les auteurs considèrent qu'il se compose principalement de l'environnement physique et de la plateforme d'interaction. Avec la proposition du *framework* CAMELEON, le cadre de référence pour l'aide à la conception des approches IDM plastiques, les même auteurs (Calvary *et al.*, 2003, 2004), ont intégré l'utilisateur dans le contexte. Désormais, le contexte d'utilisation est compose du triplet ≺Utilisateur ; Environnement ; Plateforme≻. Cette définition du contexte est la plus utilisée dans le domaine de l'IHM, notamment pour l'adaptation des interfaces.

En réfléchissant aux définitions précédentes du contexte, (Dourish, 2004) identifie quatre caractéristiques qui peuvent décrire cette notion :

1. Le contexte est une information, ce qui implique qu'il peut être connu et codé ;
2. Le contexte est délimité, ce qui implique que pour certaines applications les contextes, peuvent être prédits ;
3. Le contexte est stable, une fois les éléments du contexte identifiés, ils restent les mêmes ;
4. Le contexte et l'activité de l'utilisateur sont séparables (i.e. le contexte ne dépend pas de l'activité et c'est l'inverse qui est vrai).

En 2007, (Zimmermann *et al.*, 2007) adoptent la définition de (Dey *et al.*, 1999) et l'ont étendu en ajoutant deux nouveaux éléments. Ils considèrent que le contexte se compose d'une définition formelle qui décrit son apparence et d'une définition opérationnelle qui décrit son utilité ainsi que son comportement dynamique. Pour les auteurs, toute information décrivant le contexte d'une entité entre dans l'une des catégories suivantes : l'individualité, l'activité, la localisation, le temps, et les relations.

1. La catégorie individualité contient des propriétés et des attributs décrivant l'entité elle-même ;
2. La catégorie d'activité couvre toutes les tâches de cette entité ;
3. La localisation et le temps décrivent le cadre spatiotemporel de l'entité ;
4. Les relations représentent des informations sur toute relation possible, que peut établir l'entité avec une autre.

Le tableau suivant synthétise l'ensemble des définitions présentées précédemment. La première colonne indique le nom de l'auteur ainsi que l'année de suggestion de la proposition, et la deuxième colonne contient les éléments à prendre en compte dans le contexte :

TABLE 1.1 – Tableau récapitulatif des définitions de la notion du contexte

Auteur & Année	**Éléments définissant le contexte**
(Schilit *et al.*, 1994)	Environnement social de l'utilisateur - Localisation de l'utilisateur - Ressources à disposition de l'utilisateur - Réseau
(Brown, 1996)	Objets de l'environnement
(Brown *et al.*, 1997)	Identité de l'utilisateur - Localisation de l'utilisateur - Temps - Température - Saison
(Ryan *et al.*, 1997)	Identité de l'utilisateur - Localisation de l'utilisateur - Temps
(Ward *et al.*, 1997)	Personnes accompagnant l'utilisateur - Localisation de l'utilisateur - Ressources à disposition de l'utilisateur - Localisation des objets entourant l'utilisateur
(Pascoe, 1998)	Localisation de l'utilisateur - Objets de l'environnement - Plateforme d'interaction
(Dey *et al.*, 1999)	Utilisateur - Localisation de l'utilisateur - Objets de l'environnement - Application
(Schmidt *et al.*, 1999)	Identité de l'utilisateur - Environnement social de l'utilisateur - Activité de l'utilisateur - Localisation de l'utilisateur - Objets de l'environnement
(Thevenin et Coutaz, 1999)	Objet - Utilisateur - Évènement
(Lieberman et Selker, 2000)	Objets de l'environnement - Plateforme d'interaction
(Winograd, 2001)	Utilisateur - Localisation de l'utilisateur - Objets de l'environnement
(Calvary *et al.*, 2001)	Objets de l'environnement - Plateforme d'interaction
(Calvary *et al.*, 2004)	Utilisateur - Objets de l'environnement - Plateforme d'interaction
(Dourish, 2004)	Utilisateur - Objets de l'environnement - Plateforme d'interaction
(Zimmermann *et al.*, 2007)	Attributs de l'entité - Activité de l'entité - Localisation de l'entité, Temps, Relations de l'entité.

Comme l'a montrée le tableau, la définition du contexte ne cesse d'évoluer. De notre part et dans le cadre de cette thèse, nous adoptons celle proposée par (Calvary *et al.*, 2004). Dans ce qui suit, la mention du contexte fait référence au contexte d'utilisation défini comme suit :

Définition. *Le contexte d'utilisation dans le cadre d'un système interactif est défini par le triplet (Utilisateur, Environnement, Plateforme). L'utilisateur représente la personne qui interagit avec le système. Elle peut être caractérisée par ses préférences ainsi que ses capacités perceptuelles, cognitives et motrices. La plateforme constitue l'ensemble des*

ressources matérielles et logicielles dont dispose l'utilisateur pendant l'interaction. L'environnement représente l'ensemble des objets, personnes et événements qui peuvent avoir un impact sur le système

1.3.2 La modélisation du contexte

À travers le panel des définitions de la notion du contexte, présenté dans le chapitre 1, §1.3.1, nous constatons qu'il y a des définitions théoriques, d'autres modèlisables et opération-nelles dans un espace de travail bien déterminé.

Pour être en mesure de traiter le contexte d'une manière informatisée, un modèle de contexte est nécessaire. Étant donné l'importance de la prise en compte du contexte dans le domaine des IHM, plusieurs travaux ont essayé de modéliser ce contexte afin de le rendre opérationnel et, comme (Strang et Popien, 2004; Pessoa *et al.*, 2007; Pérez *et al.*, 2009) l'ont confirmé, il existe plusieurs méthodes de modélisation du contexte. Un bon formalisme de modélisation du contexte réduit la complexité des applications et améliore leur maintenabilité ainsi que leur évolutivité (Bettini *et al.*, 2010). En outre, une représentation formelle du contexte dans un modèle est nécessaire pour la vérification de la cohérence des éléments qui le constituent. Ces modèles de contexte sont partis d'un ensemble de représentations statiques, non-expressives vers d'autres modélisations sémantiques, extensibles et plus souples. En se basant sur les travaux de (Baldauf *et al.*, 2007; Bolchini *et al.*, 2007; Bettini *et al.*, 2010), nous identifiions les modèles suivants :

- *Le modèle attribut-valeur* : Selon (Pérez *et al.*, 2009), il s'agit de la première manière pour représenter le contexte d'une façon structurée à travers une collection de paires clé-valeur. Il s'agit du modèle le plus simple, qui décrit le contexte avec un ensemble d'attributs qui sont faciles à manipuler. Le problème de ce formalisme est qu'il n'est pas suffisant pour des systèmes complexes. Parmi les travaux qui ont opté pour cette modélisation, nous pouvons citer le travail de (Schilit *et al.*, 1994).
- *Le modèle basé sur XML* : Ce type de modèles est souvent utilisé dans une structure hiérarchique composée d'un ensemble de balises avec des attributs et de leurs contenus. (Dey, 2000) et (Han *et al.*, 2005) ont utilisé ce type de modélisation pour la mise en œuvre des modèles du contexte qu'ils ont proposés.
- *Le modèle graphique* : Il est utilisé particulièrement pour structurer le modèle d'une façon graphique, mais pas pour l'instancier. L'exemple le plus répandu de ce type de modélisation est le langage UML. Parmi les travaux qui ont adopté cette modélisation du contexte nous trouvons (Taconet et Aoul, 2008) qui a représenté le contexte sur plusieurs niveaux à l'aide du langage Ecore ainsi que (Ali *et al.*, 2008) qui proposent un diagramme de classe en UML pour décrire la localisation de l'utilisateur.
- *Le modèle orienté-objet* : Il s'agit de modéliser le contexte à travers des bases de données relationnelles. De cette façon, le contexte est stocké dans des tables et traité par les SGBD (systèmes de Gestion des Bases de Données). Cette approche est mieux adaptée pour les situations où le contexte est prédéfini à l'avance ou bien lorsque les

fonctionnalités de l'application sont limitées au domaine de la Recherche d'Information (RI) (Pérez *et al.*, 2009).

- *Le modèle logique* : Dans ce cas , le contexte est défini comme un ensemble de faits, des expressions et des règles. Ce type de modèle est caractérisé par un haut degré de formalisation et il est utilisé avec des règles spécifiques pour dériver des informations concernant le contexte ou pour détecter les inconsistances dans un modèle bien déterminé. Une des premières propositions de ce type de modèle a été proposée par (Akman et Surav, 1997) . En plus du modèle graphique qu'ils ont proposé, (Ali *et al.*, 2008) ont défini un deuxième modèle du contexte implémenté en Datalog, un langage logique basé sur des règles et des requêtes pour manipuler les données.
- *Le modèle à base d'ontologie* : Comme l'a affirmé (Pérez *et al.*, 2009), les ontologies (cf. chapitre 1, §1.4) sont utilisées pour définir les relations entre les concepts et les utiliser plus tard pour le raisonnement. Un modèle de contexte sous forme d'ontologie se compose essentiellement d'une base de connaissances qui représente les concepts du contexte et les relations entre eux. Parmi les premières initiatives introduisant ce type de modélisation, figure celle présentée par (Oztürk et Aamodt, 1997).

1.3.3 La sensibilité au contexte

Après avoir trouvé une définition du contexte, nous allons présenter la notion de la sensibilité tel qu'elle a été définie dans le domaine de l'IHM. En effet, cette notion apparue en 1994, a été introduite par (Schilit *et al.*, 1994) comme étant la capacité à s'adapter en fonction de la localisation, l'ensemble de personnes à proximité, les plateformes accessibles...

En 1998, Pascoe (1998) a défini la sensibilité au contexte comme étant la capacité des dispositifs informatiques à détecter, interpréter et répondre aux aspects de l'environnement local d'un utilisateur et les appareils informatiques eux mêmes.

(Dey et Abowd, 2000) ont affiné ces deux définitions en une autre plus générale. Ils affirment qu'un système est sensible au contexte s'il utilise le contexte pour fournir à l'utilisateur des informations et des services pertinents. Les auteurs distinguent trois catégories de sensibilité au contexte :

1. La présentation des informations et des services ;
2. L'exécution automatique d'un service ;
3. Le stockage d'information selon le contexte.

Les systèmes sensibles au contexte peuvent être implementes de différentes manières. Dans le but de simplifier leurs développements et leurs déploiements et afin d'améliorer l'extensibilité et la réutilisabilité des systèmes, une séparation entre la détection et l'utilisation du contexte est nécessaire. L'architecture en couche présentée dans la Figure 1.7, augmente les couches de détection et d'utilisation, en ajoutant les fonctionnalités d'interprétation et de raisonnement.

FIGURE 1.7 – Architecture générale d'un système sensible au contexte

Cette architecture conceptuelle a été proposée par (Dey *et al.*, 2001) et constitue un modèle suivi par la majorité des travaux dans le domaine de la sensibilité au contexte. Elle se compose principalement de cinq couches :

- *La capture du contexte* : Elle se compose d'un ensemble de différents capteurs. Il est à noter que le capteur ne représente pas uniquement le matériel de détection, mais aussi toutes les sources de données qui peuvent fournir des informations sur le contexte.
- *L'extraction des données* : La seconde couche est responsable de l'extraction des premières données du contexte. Cette étape nécessite l'utilisation des pilotes et des API (Application Programming Interface) appropries accompagnant les capteurs.
- *Le prétraitement du contexte* : Elle permet d'interpréter les données acquises depuis la couche précédente et de faciliter leurs utilisation, en les transformant en d'autres formats plus faciles à exploiter.
- *Le stockage du contexte* : Cette couche permet d'organiser les données recueillies et de les stocker pour d'autres utilisations ultérieures. Pour pouvoir stocker ces informations, un modèle permettant de décrire le contexte est nécessaire. La notion de modélisation du contexte ainsi qu'un panel des principaux formalismes de modélisation du contexte ont été présentés dans le chapitre 1, §1.3.2.
- *L'application* : Elle représente le système final, qui exploite les données issues de la couche inférieure pour fournir une application sensible au contexte. C'est au niveau de cette couche que s'effectue l'implémentation de l'ensemble de réactions aux différents évènements qui peuvent se produire.

Face à la diversité des sources d'informations de plus en plus nombreuses et complexes et afin d'introduire des descriptions explicites des contenus, les technologies sémantiques utilisent plusieurs manières de représentations des connaissances. Parmi les plus récentes, figure l'ontologie. Le section suivante sera consacrée à l'introduction de cette notion.

1.4 Les ontologies dans la conception des applications interactives

Les ontologies permettent de représenter formellement les connaissances, de décrire le raisonnement sur ces connaissances et de les partager et les réutiliser par plusieurs applications. Les ontologies sont utilisées dans de nombreux domaines. En se basant sur (Guarino, 1998; Berners-Lee *et al.*, 2001), nous citons les domaines de l'ingénierie des connaissances, l'ingénierie des langages, la conception des bases de données, la recherche d'information, et dernièrement elles sont exploitées dans le domaine de E-commerce et dans le domaine du Web sémantique.

1.4.1 Définitions

Le terme d'ontologie était déjà utilisé en philosophie depuis le XIX$^{\text{ème}}$ siècle pour designer l'étude des propriétés générales de ce qui existe. En informatique, il existe plusieurs définitions de ce terme. La première a été proposée par (Neches *et al.*, 1991) qui la définissent comme étant l'ensemble des termes et des relations comportant le vocabulaire d'un domaine ainsi que les règles combinant ces termes et ces relations pour définir les extensions de ce vocabulaire.

La définition la plus convenue de l'ontologie a été proposée par (Gruber, 1993) dans laquelle il considère l'ontologie comme étant une spécification formelle et explicite d'une conceptualisation partagée. La *conceptualisation* se réfère à un modèle abstrait des concepts pertinents caractérisant des phénomènes dans le monde. *"Explicite"* précise que le type de concepts utilisés et les contraintes sur leur utilisation sont explicitement définis. *"Formelle"* insiste sur le fait que l'ontologie doit être compréhensible par la machine. *"Partagée"* reflète qu'une ontologie doit capturer la connaissance consensuelle acceptée par les différentes communautés.

(Guarino et Giaretta, 1995) affinent la définition précédente en qualifiant la spécification des ontologies de partielle étant donné que la formalisation de la conceptualisation est dépendante de la capacité limitée des langages permettant d'implémenter les ontologies.

Dans ce mémoire, nous adoptons la définition proposée par le W3C [3] :

Définition. *L'ontologie définit les termes utilisés pour décrire et représenter un domaine de connaissance. Elle comprend les définitions des concepts de base dans un domaine ainsi que les relations entre eux.*

3. http ://www.w3.org/

De cette définition, il en découle qu'une ontologie est composée du 5-uplets ≺C ; I ; R ; P ; A ≻ où :

- C est un ensemble de **concepts** ou de classes qui permettent de définir des collections d'individus. Les concepts sont généralement présentés dans une organisation hiérarchique.
- I est un ensemble d'**instances** qui sont des objets concrets d'une classe.
- R est un ensemble de **relations** qui permettent de définir les liens entre les classes.
- P est un ensemble de **propriétés** ou d'attributs qui permettent de caractériser les classes et leurs instances
- A est un ensemble d'**axiomes** permettant de spécifier les restrictions sur les relations entre les classes.

La Figure 1.8 représente un exemple graphique d'une ontologie. Celle-ci est composée d'une collection de classes, leurs propriétés et l'ensemble de restrictions qui décrivent le domaine des Pizzas. A droite de la figure, nous avons un groupe d'instances possibles de ces classes.

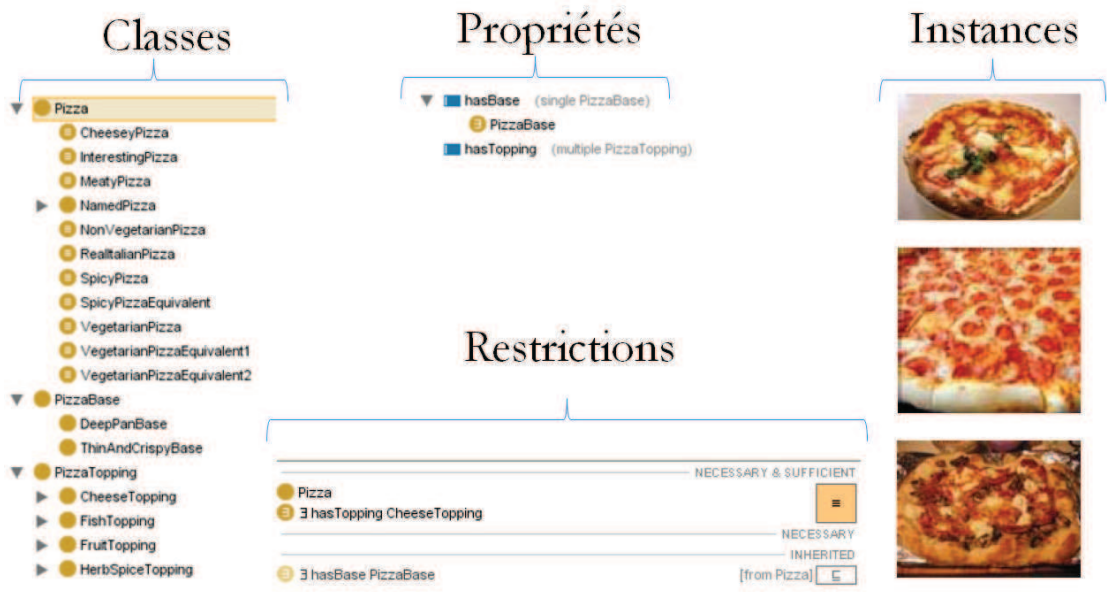


FIGURE 1.8 – Représentation graphique d'une ontologie en utilisant l'outil Protégé

1.4.2 Classification des ontologies

Les ontologies présentent différents niveaux d'abstraction dans les détails des conceptualisations capturées, ce qui entraine une variété dans leurs classifications. Cette différence de classification est dû également aux critères pris en compte pendant cette classification. Parmi les classifications les plus connues, nous pouvons souligner celles de (van Heijst

et al., 1997; Guarino, 1997; Lassila et McGuinness, 2001; Gómez-Pérez *et al.*, 2004). En se basant sur ces travaux, nous pouvons identifier les groupes d'ontologies suivants :

– *Les ontologies de représentation des connaissances* : elles fournissent des éléments de modélisation primitifs pour caractériser les connaissances.
– *Les ontologies génériques* : elles représentent des connaissances communes qui peuvent être utilisées dans différents domaines.
– *Les ontologies de haut niveau* : elles représentent des éléments du monde réel d'une manière générique et avec une haute abstraction.
– *Les ontologies du domaine* : elles permettent de décrire un domaine specique (la médecine, le droit, le transport, etc.). La plupart des ontologies existantes sont des ontologies du domaine.
– *Les ontologies des tâches* : elles décrivent le vocabulaire lié à une tâche ou une activité (objectifs, des calendriers, etc.).
– *Les ontologies de domaine-tâches* : ce sont des ontologies des tâches qui peuvent être réutilisées dans un domaine spécifique, mais pas de façon générique dans des domaines similaires.
– *Les ontologies de méthode* : elles fournissant des définitions pertinentes des concepts et des relations qui décrivent la réalisation d'une tâche particulière.

1.4.3 Langages de description et outils de manipulation des ontologies

Afin de représenter et de gérer les connaissances sémantiques, les technologies basées sur les ontologies contiennent : les langages de description de l'ontologie, les parseurs de l'ontologie, les langages de requête, ainsi que les environnements de développement et de gestion de l'ontologie. Dans la suite, nous présenterons un panel des principaux langages de description des ontologies ainsi que l'ensemble des outils de leurs éditions.

1.4.3.1 Les langages de description des ontologies

D'un point de vue opérationnel, nous pouvons bâtir une ontologie grâce aux langages classiques de la programmation logique (par exemple : Prolog, Lisp, etc). Mais, plus souvent, nous utilisons des modèles et des langages spécialisés pour la construction d'ontologies. Les langages de description d'ontologies fournissent les moyens de représenter formellement les connaissances sémantiques. Le modèle en couches, décrit dans la Figure 1.9, contient une illustration de la hiérarchie des langages de description de l'ontologie, où chaque couche exploite et utilise les capacités des couches inférieures.

- XML (Extensible Markup Language) (Bradley, 1998) est un langage à base de balises qui permet une semi-structuration des documents. Chaque document XML est conforme à son XML-schema qui définit sa structure mais qui n'impose aucune contrainte au niveau sémantique.

- SHOE (Simple HTML Ontology Extensions) (Luke et Heflin, 2000) est une extension des balises HTML (Hyper Text Markup Language) qui permet de rajouter de la sémantique dans ce type de documents.
- RDF (Resource Description Framework) (W3C, 1999) est un langage qui permet de créer un modèle de données (appelé méta-donnée) pour les ressources et ainsi que pour les associations qui les relient. Il permet de représenter l'information sous la forme d'un graphe.
- RDFS (Resource Description Framework Schema) (Brickley et Guha, 2004) fournit un vocabulaire de base pour décrire les propriétés et les classes des ressources RDF. En utilisant RDFS, il est possible de créer des hiérarchies de classes et de propriétés.
- OIL (Ontology Inference Layer) (Fensel *et al.*, 2000) est à la fois un langage de représentation et d'échange pour les ontologies. Il combine les primitives des langages reposant sur les frames avec une sémantique formelle et des possibilités de raisonnement issues de la logique de description.
- DAML+OIL (van Harmelen *et al.*, 2001) est un langage de balisage sémantique pour décrire les ressources Web. Il s'appuie sur les normes RDF et RDF Schema et les étend pour avoir plus de richesse dans la modélisation.
- XOL (XML based Ontology Exchange Language) (Karp *et al.*, 1999) est un langage basé sur XML qui a été créé pour satisfaire les objectifs du groupe BioOntology réclamant la nécessité d'un langage sémantique permettant la représentation des connaissances dans le domaine de la Bio informatique. Ce langage permet principalement l'échange des ontologies décrivant ce domaine mais il peut être appliqué à d'autres disciplines.
- OWL (Web Ontology Language) (McGuinness et Van Harmelen, 2004) est un standard sémantique basé sur XML qui permet de publier et partager des ontologies sur le Web. OWL, approuvé par le W3C, a été développé comme une extension de RDF et DAML + OIL. Il permet par exemple de rajouter des contraintes supplémentaires, au niveau des cardinalités, ou bien au niveau des caractéristiques des propriétés telle la transitivité. Afin de permettre une meilleure utilisation du langage, celui-ci a été défini suivant trois parties : OWL-Lite, OWL-DL et OWL-Full.

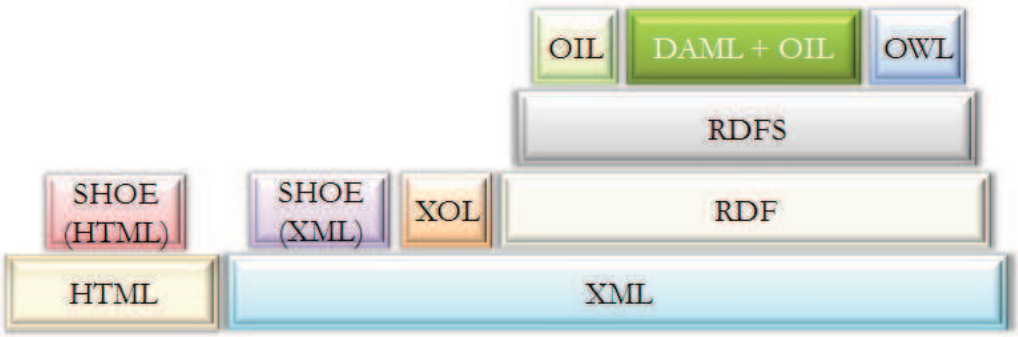


FIGURE 1.9 – Langages de description des ontologies (adaptée de (W3C, 2004))

1.4.3.2 Outils d'édition des ontologies

De nombreux outils de construction d'ontologies utilisent des formalismes variés et offrent différentes fonctionnalités. Tous ces outils offrent des supports pour le processus de création d'ontologies, mais peu offrent une aide à la conceptualisation. Parmi ces outils, nous citons les trois suivants :

- Protégé [4] (Noy *et al.*, 2001) est l'un des outils de développement des ontologies les plus utilisés, qui a été développé à l'Université de Stanford. Protégé est un outil gratuit et open source qui fournit un éditeur intuitif pour créer les ontologies et il est doté d'un ensemble d'extensions permettant la visualisation des ontologies sous formes de graphes, la gestion des projets, etc.
- Ontolingua [5] est un serveur localisé à l'Université de Stanford qui permet le développement des ontologies avec une interface Web basée sur un formulaire. L'application principale que contient le serveur est un éditeur d'ontologies qui permet de créer des ontologies même d'une maniéré collaborative et de les visualiser.
- OntoEdit (Sure *et al.*, 2002) a été développé par les groupe de gestion des connaissances à l'université de Karlsruhe. Il s'agit d'un environnement d'Ingénierie des ontologies qui permet la création, la navigation, l'entretien et la gestion des ontologies. L'environnement prend en charge le développement collaboratif des ontologies. Ceci est archivé par son architecture client / serveur où les ontologies sont gérées dans un serveur central et divers clients peuvent accéder et modifier ces ontologies.

1.5 La modélisation des tâches

1.5.1 Définitions

L'analyse des tâches est largement reconnue comme un moyen fondamental pour améliorer la manière avec laquelle un utilisateur peut interagir avec une IHM, dans le but d'accomplir une tâche interactive donnée (Hackos et Redish, 1998). D'après (Storrs, 1995), une tâche peut être définie comme un objectif qui serait satisfaisant dans un contexte approprié et accompagné d'un ensemble d'autres tâches.

Les méthodes d'analyse des tâches sont utilisées pour analyser et décrire une tâche particulière de l'utilisateur. Ces méthodes ont bénéficié de l'expertise de plusieurs disciplines telles que :

- Les sciences cognitives (Van Der Veer *et al.*, 1996) ;
- L'ethnographie (Jordan, 1996) ;
- Le génie logiciel (Smith et O'Neill, 1996) ;

4. http ://protege.stanford.edu/
5. http ://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/reference-manual/index.html

- L'Interaction Homme-Machine (Paterno, 1999).

Dans le cadre de ce mémoire et dans ce qui suit, nous ne nous intéresserons qu'à la modalisation des tâches dans le domaine de l'IHM. De ce fait, nous adopterons la définition proposée par (García *et al.*, 2008) :

Définition. *Les modèles des tâches sont des descriptions logiques des activités qui sont conçues pour être realisées afin d'atteindre les objectifs de l'utilisateur dans un système interactif. Ils sont généralement utilisés lors de l'analyse d'un domaine et ils pourront être utilisés comme un guide lors de la conception des systèmes.*

(Paterno, 2002) affirme que, dans le cadre de l'IHM, la modélisation des tâches peut être utilisée pour :

- Comprendre un domaine d'application.
- Documenter un logiciel interactif.
- Concevoir de nouvelles applications compatibles avec le modèle de l'utilisateur.
- Analyser et évaluer l'utilisabilité d'un système interactif.

1.5.2 Le processus métier au service de la modélisation des tâches

Mis à part le modèle de tâche et dans une autre discipline, nous trouvons le modèle de *workflow*. Dans la littérature, les termes "processus" et "modélisation de *workflow*" sont parfois mélangés pour décrire une séquence d'activités à remplir pour atteindre un but (Pontico *et al.*, 2006). Ce mélange est dû au fait que les concepteurs de *workflow* sont amenés à inclure les activités humaines dans leurs modèles, et non seulement des activités automatisées. Un *workflow* se caractérise en fonction du processus métier. Dans le cadre de notre thèse, pour définir ces deux notions nous adoptons les spécifications proposées par (WMCS, 1999) :

Définition. *Le workfow est l'automatisation, totale ou partielle, d'un processus métier au cours de laquelle des documents, des informations ou des tâches sont transmis selon un ensemble de règles.*

Définition. *Un processus métier est un ensemble de procédures ou d'activités qui sont liées, afin de réaliser un objectif, dans le cadre d'une structure organisationnelle définissant les rôles et les relations fonctionnelles.*

Certains auteurs (Trætteberg, 1999; Eichholz *et al.*, 2005; Kristiansen et Trætteberg, 2007; García *et al.*, 2008) considèrent que la modélisation des tâches et la modélisation des *workflows* font parties du même domaine et traitent les mêmes problèmes mais à différents niveaux d'abstraction. En effet, ces deux formalismes décrivent comment les tâches et les activités seront coordonnées, de telle sorte que leur exécution accomplit les objectifs métiers.

(Trætteberg, 2002) affirme que les modèles du *workflow* possèdent une granularité forte car ils décrivent les activités métiers avec des détails d'exécution. Il affirme également que le modèle des tâches recouvre partiellement le modèle du *workflow*. Ce chevauchement est dû au fait que le niveau le plus bas du modèle de *workflow* est généralement formé par des tâches effectuées par un utilisateur unique. En effet, les modèles du *workflow* se concentrent principalement sur la description du travail collaboratif, tandis que les modèles des tâches représentent principalement l'exécution de la tâche individuellement (Eichholz *et al.*, 2005).

Le Tableau 1.2 présente une comparaison entre la modélisation des *workflows* et celle des tâches. Afin d'identifier les critères de cette comparaison, nous nous sommes basés sur nos besoins ainsi que sur les travaux précédemment présentés (Trætteberg, 1999, 2002; Eichholz *et al.*, 2005; Kristiansen et Trætteberg, 2007; García *et al.*, 2008). Ces critères se résument comme suit :

- *Le fond de la recherche* - le domaine de recherche duquel est issu le modèle ;
- *L'objectif global* - le but global qui permet d'atteindre le modèle ;
- *Le centre d'intérêt spécifique* - le but spécifique du modèle ;
- *La granularité du modèle* - le niveau d'abstraction du modèle ;
- *Les concepts traités* - les éléments de base constituant le modèle.

TABLE 1.2 – Différence entre la modélisation des tâches et la modélisation du workflow

Aspect / Type du modèle	**Modèle des tâches**	**Modèle du workflow**
Le fond de la recherche	Analyse de la tâche humaine	La théorie des organisations
L'objectif global	Définir la manière avec laquelle les tâches et les activités sont coordonnées pour accomplir les objectifs métiers.	
Centre d'intérêt spécifique	Décrire les objectifs individuels d'un utilisateur particulier	Décrire l'organisation d'un travail collaboratif
Granularité	Niveau de granularité faible	Niveau de granularité élevé
Les concepts traités	Les mêmes concepts sont utilisés dans chaque modèle avec des significations différentes	

1.5.3 Formalismes de la modélisation des tâches

Dans le cadre des IHM, un certain nombre de notations de modélisation des tâches ont été développés, souvent avec des objectifs et des atouts différents. Par exemple, chaque notation est généralement conçue pour être employée à une phase spécifique du cycle de vie

du développement logiciel ou pour la conception d'un type particulier de système (Balbo *et al.*, 2004). Sans prétendre à l'exhaustivité, nous présentons dans ce qui suit quelques modèles représentatifs de l'analyse et la modélisation des tâches dans le cadre de l'IHM. Pour plus de détails sur les différentes notations de la modélisation des tâches, le lecteur peut se référer aux travaux de (Limbourg et Vanderdonckt, 2003), (Balbo *et al.*, 2004) et (Pontico *et al.*, 2006).

HTA (Hierarchical Task Analysis)

HTA (Hierarchical Task Analysis) (Annett et Duncan, 1967) , nommé l'ancêtre des modèles, permet de décrire la manière avec laquelle un travail est organisé afin de répondre à un objectif bien déterminé. Il s'agit d'identifier l'objectif visé, puis les diverses sous-tâches et les conditions qui doivent être menées pour atteindre cet objectif. De cette façon, les tâches complexes peuvent être représentées comme une hiérarchie des tâches ou une tâche peut être décomposée en un ensemble de sous-tâches, si nécessaire, et de plans (les conditions qui sont nécessaires pour bien mener le séquencement des sous-tâches). Pour spécifier l'ordre dans lequel les sous-tâches d'une tâche donnée peuvent être exercées, HTA fournit la possibilité d'utiliser les plans. Un plan est nécessaire pour chaque niveau hiérarchique agissant ainsi comme une condition sur l'exécution des tâches.

GOMS (Goals, Operators, Methods, Selectors)

GOMS (Goals, Operators, Methods, Selectors) (Card *et al.*, 1983) fournit un langage de haut niveau pour l'analyse des tâches et la modélisation des IHM. Cette analyse repose sur la description des activités de l'utilisateur selon :

1. *Les buts* qui représentent ce que l'utilisateur souhaite obtenir. Chaque but peut être décomposé en sous-buts ;
2. *Les opérateurs* qui sont des opérations élémentaires réalisées au service d'un but spécifique ;
3. *Les méthodes* représentant une séquence d'opérateurs qui accomplissent un but ;
4. *Les règles de sélection* qui justifient le choix d'une méthode parmi plusieurs, pour atteindre ce même but.

Un modèle en GOMS permet de décrire le choix des méthodes et des opérateurs nécessaires pour simuler le comportement de l'utilisateur. Il existe différentes variantes de GOMS qui utilisent des termes différents et qui fonctionnent à différents niveaux d'abstraction. Nous pouvons citer KLM (Keystroke-Level Model) (Card *et al.*, 1980), CPM-GOMS (Gray *et al.*, 1993)et NGOMSL (Kieras, 1997).

CTT (ConcurTaskTrees)

CTT (Concurrent Task Trees) (Paterno, 1999) est une notation largement utilisée pour la modélisation des tâches. L'auteur a défini quatre types de tâches : les *tâches utilisateurs* qui pourraient être cognitives/perceptives, les *tâches d'interaction* qui représentent une interaction de l'utilisateur avec le système, les *tâches d'application* qui sont exécutées par le système, et les *tâches d'abstraction* qui se réfèrent à des tâches complexes.

Un modèle de tâches défini en CTT est représenté sous forme d'arborescence. Les relations entre les tâches sont de deux types : soit *hiérarchiques* définies entre tâches mères et tâches filles, soit *temporelles* définies entre tâches sœurs voisines. Pour définir les relations temporelles entre les tâches, CTT offre un nombre important d'opérateurs, issus principalement de la notation LOTOS (Language Of Temporal Ordering Specification) (ISO, 1989). Ces opérateurs seront détaillés avec des exemples qui illustrent leurs utilisations dans l'annexe A1.

La notation CTT est modifiable et analysable grâce à l'outil CTTE (ConcurTaskTrees Environment) [6] (Mori *et al.*, 2002), qui permet de générer des scénarios utiles pour l'analyse des tâches et la simulation de l'utilisation de l'application.

BPMN (Business Process Modeling Notation)

BPMN (Business Process Modeling Notation) est un standard pour la modélisation des processus métiers qui fournit une notation graphique basée sur un ensemble d'éléments simples, faciles à utiliser et accessibles à tous les experts métiers. La première version de BPMN a été développée par le Business Process Management Initiative (BPMI, 2004), et elle est actuellement maintenue par l'OMG (Object Management Group) depuis leur fusion en 2005. La version actuelle de BPMN, sortie en Mars 2011, est 2.0 [7] et elle a pour but d'offrir une notation explicite, et facile d'emploi se basant sur un ensemble d'éléments graphiques simples qui sont repartis sur trois types :

- *Les conteneurs* (perspective organisationnelle) qui sont des partitions du processus représentant un acteur ou d'une entité organisationnelle particulière.
- *Les nœuds* (perspective fonctionnelle) représentent les activités, les évènements et les branchements conditionnels.
- *Les arcs* (perspective informationnelle) représentent les flux d'information. Il en existe trois types : les flux de contrôle séquentiel qui caractérisent une communication interne ; les flux de message qui caractérisent une communication inter-conteneurs ; les associations. La notation BPMN sera détaillée dans l'annexe A1.

Bien que le formalisme BPMN est essentiellement créé pour la modélisation des processus métier, certains auteurs l'ont proposé comme un langage pour la modélisation des tâches

6. http ://giove.cnuce.cnr.it/ctte.html
7. http ://www.omg.org/spec/BPMN/2.0/

(Trætteberg, 1999; Bouchelligua *et al.*, 2010; Brossard *et al.*, 2011). Ils considèrent que les processus métier et les tâches d'interaction sont des concepts similaires qui ont beaucoup de caractéristiques communes, mais qui sont présentés à différents niveaux d'abstraction. En raison de ces chevauchements considérables, ils proposent d'étendre BPMN pour modéliser les tâches.

1.5.4 Récapitulatif et synthèse

Après avoir présenté quelques notations permettant la modélisation des tâches dans le cadre des IHM, nous proposons une étude comparative entre ces formalismes. Cette étude, illustrée par le Tableau 1.3, se base sur les critères suivants :

- *Une notation standardisée* ? - Pour indiquer si la notation est considérée comme étant un standard dans le domaine de la modélisation des tâches ;
- *Présentation de la hiérarchie des tâches* ? - Pour savoir si la notation fournit la possibilité de décomposition hiérarchique des tâches ;
- *Richesse en opérateurs* - Présente l'ensemble d'éléments que fournit la notation pour modéliser les tâches ;
- *Spécification de l'acteur de la tâche* ? - Révèle si la notation propose la possibilité d'associer chaque tâche à un acteur ;
- *Prise en compte du domaine d'application* ? - Spécifie si le modèle permet de prendre en compte les concepts du domaine d'application ;
- *Prise en compte du contexte* ? - Désigne si le modèle proposé fournit la possibilité de prendre en compte les éléments du contexte ;
- *Possibilité d'extension de la notation* ? - Permet de connaitre si la notation est capable d'être étendue pour un but spécifique ;
- *Représentation du passage du flux d'information* ? - Indique si le modèle permet de supporter le passage de flux d'information entre les différents éléments du modèle des tâches ; et
- *Représentation de la dynamique de l'application* ? - Pour savoir si la notation permet de traduire le comportement de l'application et simuler l'interaction avec l'utilisateur final pendant le runtime.

TABLE 1.3 – Tableau comparatif des modèles de tâches présentés

Critères / Notation	**HTA**	**GOMS**	**CTT**	**BPMN**
Une notation standardisée ?	Non	Non	Non	Oui
Présentation de la hiérarchie des tâches ?	Oui	Oui	Oui	Oui
Richesse en opérateurs	Nombre limité d'opérateurs	Nombre limité d'opérateurs	- Quatre types de tâches - Huit opérateurs	- Conteneurs - Nœuds - Arcs
Spécification de l'acteur de la tâche ?	Non	Non	Non (permet d'indiquer qu'il y a une interaction mais ne permet pas de spécifier l'acteur)	Oui
Prise en compte du domaine ?	Non	Non	Oui (annotation par des concepts du domaine)	Non
Prise en compte du contexte ?	Non	Non	Non	Non
Possibilité d'extension de la notation ?	Non	Non	Oui	Oui
Représentation du passage du flux d'information ?	Non	Non	Oui (uniquement entre tâches voisines)	Oui
Présentation de la dynamique de l'application	Non	Non	Non	Oui

A travers le tableau précédent, nous pouvons déduire ce qui suit :

- Les modèles des tâches peuvent être représentés aux différents niveaux d'abstraction et la plupart d'entre eux permettent de modéliser la structure de chaque tâche et sa décomposition en sous-tâches. En effet, lorsque les concepteurs ont l'intention de fournir des orientations spécifiques à la conception, les activités sont représentées à un niveau de granularité bas, sinon s'ils veulent préciser les exigences sur la manière avec laquelle les activités doivent être effectuées, ils ne considèrent que les tâches de haut niveau (Vanderdonckt *et al.*, 1998).
- La plupart de ces notations permettent de décrire le rôle de chaque tâche et l'ordonnancement temporel entre elles où chaque notation utilise son propre formalisme syntaxique et sémantique. La facilité et l'efficacité de cet ordonnancement dépendent fortement de la richesse de la notation, en termes d'opérateurs. En effet, plus le modèle est riche en opérateurs, plus il est facile au concepteur d'exprimer les besoins du système.
- Étant donné la diversité des domaines et des exigences, il est important qu'un modèle des tâches soit extensible. Cette extensibilité assure à la notation, la continuité de son utilisation et permet au concepteur de l'adapter selon ses convenances. (Limbourg et Vanderdonckt, 2003) affirment que chaque modèle est adapté à un contexte déterminé et qu'il n'existe pas un modèle des tâches universel. Parmi les modèles présentés précédemment, cette option n'est fournie que par BPMN et CTT. Il existe de nombreux travaux qui ont enrichi BPMN, nous trouvons (Brossard, 2008; Bouchelligua *et al.*, 2010) et CTT.
- Dans un modèle des tâches chaque tâche peut avoir certains nombre d'attributs tels que l'acteur qui l'exerce ou bien les ressources qu'elle utilise, etc. Les auteurs du cadre de référence CAMELEON (Calvary *et al.*, 2003) proposent de considérer également les concepts du domaine (appartenant à un modèle de domaine) lors de la conception du modèle des tâches. CTT permet d'annoter chaque tâche par le concept du domaine qu'elle manipule. Cette possibilité étant absente dans les autres modèles, elle peut être rattrapée dans le cas de BPMN en étendant son métamodèle.
- Selon (Clerckx *et al.*, 2005), ni les modèles des tâches ni les modèles des processus métiers ne permettent la prise en compte totale du contexte, lors de la modélisation des applications. Nous remarquons, à travers le Tableau 1.3 que HTA, GOMS et BPMN ne considèrent pas le contexte. Bien que CTT fournisse la possibilité de prendre en compte la plateforme dans le modèle des tâches, il est important de considérer également les autres éléments du contexte tel qu'il a été défini par (Calvary *et al.*, 2003), à savoir l'utilisateur et son environnement.
- Si la plupart des modèles de tâches présentés auparavant permettent de décrire les interactions entre l'utilisateur et le système, rares sont ceux qui permettent la modélisation du passage du flux d'information notamment lorsque les tâches à considérer ne sont pas directement interprétables au niveau informatique comme, par exemple, un échange d'informations verbales entre deux personnes (Brossard *et al.*,

2007). À notre connaissance, l'unique modèle des tâches qui permet de prendre en compte cet aspect, est BPMN et bien que CTT permettent le passage d'information entre les tâches voisines, cette option reste limitée pour modéliser l'intégralité du passage du flux dans une application.

1.6 Les langages de description des IHM

1.6.1 Définitions

L'implémentation des IHM est une partie pertinente du développement logiciel. De nos jours, lorsque les applications doivent être mises en œuvre sur plusieurs plates-formes, le travail nécessaire pour obtenir des interfaces de haute qualité est augmentée, étant donne la variété des langages permettant de le faire, où chacun d'entre eux diffère par sa propre syntaxe et par son niveau abstraction. Cet effort peut être évité au moyen de l'IDM permettant aux concepteurs de modéliser les interfaces utilisateur à travers la spécification d'un ensemble de modèles déclaratifs d'un haut niveau d'abstraction. Afin d'obtenir une interface répondant à toutes les exigences, le concepteur est appelé à décrire son interface à travers un Langage de Description de l'Interface Utilisateur (LDIU). Dans le cadre de cette thèse, nous avons retenu la définition d'un LDIU proposée par (Souchon et Vanderdonckt, 2003) :

Définition. *Un langage de description de l'interface utilisateur est un langage formel de haut niveau, permettant de décrire une interface utilisateur particulière. Un tel langage implique la définition d'une syntaxe et d'une sémantique. Ce langage peut être considéré comme une manière commune de spécifier une interface utilisateur, indépendamment de tout langage cible qui servirait à mettre en œuvre cette interface.*

Les LDIU peuvent combler l'écart entre les modèles formels d'interactions homme-machine et des implémentations concrètes d'interfaces utilisateur (IU) (Abrams *et al.*, 1999). De ce fait, ils peuvent être utilisés à n'importe quelle étape du processus de conception. Selon (Calvary *et al.*, 2011), un LDIU peut être utilisé pendant :
– L'analyse des besoins : dans le but de recueillir et de susciter des exigences ;
– L'analyse des systèmes : dans le but d'exprimer les spécifications répondant à ces exigences ;
– La conception du système : dans le but d'affiner les spécifications en fonction du contexte ;
– L'exécution : afin de réaliser une interface utilisateur via un moteur de rendu.

1.6.2 Présentation des langages

Depuis l'avènement de la World Wide Web dans les années 90 et l'émergence de XML comme méta-langage, un certain nombre de LDIU ont vu le jour. Grâce à la flexibilité et

l'interopérabilité fournit par XML, les efforts ont été concentrés autour de ce langage et aujourd'hui, plus d'une quarantaine, de LDIUs existent. Dans ce qui suit, nous nous contentons de présenter six langages basés sur XML, où nous commençons par une présentation individuelle de chacun et nous terminons par une étude comparative. Pour plus de détails sur les LDIU, le lecteur peut se référer aux travaux de (Souchon et Vanderdonckt, 2003; Trewin *et al.*, 2004; García *et al.*, 2009).

XIML

Le eXtensible Markup Language Interface (XIML) [8] (Eisenstein *et al.*, 2000, 2001) est un langage basé sur XML qui spécifie les différents aspects de l'interaction. Une représentation d'interface utilisateur dans XIML est une collection d'éléments, dont chacun peut être classé sous différents composants. Il existe cinq composants prédéfinis et le langage n'impose pas une limite sur le nombre et le type de composants qui peuvent être définis. Ces composants sont les tâches (définissant les tâches utilisateur ainsi que le processus métier que supporte l'interface) ; le domaine (décrivant tous les objets utilisés) ; l'utilisateur (spécifiant les caractéristiques des utilisateurs pouvant interagir avec l'interface) ; le dialogue (décrivant l'interaction entre l'utilisateur et l'interface) ; et la présentation (caractérisant l'aspect statique de l'interface). Chaque composant peut être décrit par un ensemble d'attributs et de relations.

XIML supporte le développement des interfaces utilisateurs multi-plateforme, en permettant la définition de plusieurs composants de présentation au sein d'une seule spécification. Chaque composant de présentation est orienté vers une plate-forme cible et inclut la spécification des widgets, intéracteurs et des contrôles pour cette plateforme, ce qui permet la création de plusieurs interfaces pour plusieurs plateformes.

XForms

XForms (W3C, 2003, 2009b) est un standard du World Wide Web Consortium spécifique à la description des formulaires pour le Web. L'un des objectifs dans la conception de la norme XForms est de soutenir l'indépendance du code, de la plateforme. Ce langage divise les formulaires en trois parties : le modèle des données ; les instances des données ; l'interface utilisateur. Le modèle des données, à travers une représentation abstraite, décrit le but du formulaire. Les instances des données fournissent un moyen pour recueillir les données saisies par l'utilisateur, dans un format XML. Ces données sont extraites en ligne ou d'un document XML. L'interface utilisateur, représentée par les contrôles du formulaire, est indépendante de toute plateforme. Ces contrôles pourraient être interprétés de différentes façons sur différentes plates-formes. A travers la propositions de ces trois parties, XForms sépare la présentation, des données, afin de pouvoir mettre en œuvre le patron MVC (Model View Controller). Cependant, XForms n'est pas un type de document

8. http ://www. ximl.org

autonome, mais il est destiné à être intégré dans d'autres langages basés sur XML. En plus, il est limité à la modélisation des formulaires.

AUIML

Le Abstract User Interface Markup Language (AUIML) (Azevedo *et al.*, 2000; Merrick *et al.*, 2004) est un autre langage basé sur XML, qui a été conçu pour capturer l'intention de l'utilisateur. Étant donné que ce langage implémente le patron MVC, l'idée principale était d'enregistrer la sémantique d'interaction ainsi que l'intention de l'utilisateur sans se soucier des détails concernant la plateforme. Dans AUIML, une interface utilisateur est décrite en termes de trois parties : un modèle de données, un modèle d'interaction et un modèle de présentation. Le modèle de données comprend divers types de données et structures de données à recueillir ainsi que les données de sortie. Certains d'entre eux comprennent les arbres, les tableaux... L'interaction avec l'interface utilisateur est représentée à travers un ensemble d'évènements ou d'actions tandis que la présentation comprend des propriétés qui sont liées à des éléments d'interaction.

AUIML fournit une certaine assistance aux concepteurs en fournissant l'outil AUIML Toolkit, soutenu par IBMWebSphere ainsi qu'un plugin pour l'environnement de développement Eclipse [9]. Des moteurs de rendu pour Java Swing et HTML sont également disponibles pour ce langage.

AAIML

Le comité technique V2 du Comité International pour la Standardisation des Technologies d'Information (INCITS [10]) a développé le langage Alternate Abstract Interface Markup Language (AAIML) (Zimmermann *et al.*, 2003), destiné à définir une interface utilisateur spécifique pour une plateforme bien déterminée. Le langage à été proposé de façon à ce qu'il soit assez abstrait afin de pouvoir générer une description d'une interface utilisateur en se basant sur la plateforme cible.

AAIML définit un ensemble d'éléments d'interface (appelés "*interacteurs abstraits*") permettant de spécifier les opérations d'entrée et de sortie. Chaque interacteur abstrait est mappé à un "*interacteur concret*". AAIML fournit également un modèle d'évènements pour faciliter la description des interfaces utilisateurs interactives.

UsiXML

Le USer Interface eXtensible Markup Language (UsiXML) (Limbourg et Vanderdonckt, 2004; UsiXML, 2007) est l'un des LDIU bases sur XML, les plus récents, destiné à créer des

9. http ://www.icewalkers.com/Linux/Software/524910/AUIML-Toolkit.html
10. http ://www.incits.org/tc_home/v2.htm

interfaces pour différents types de plateformes (interfaces graphiques, interfaces sonores et des interfaces multimodales). Ce langage propose un ensemble de modèles pour différents niveaux d'abstractions à savoir le modèle des tâches, le modèle du domaine, le modèle de présentation, le modèle du dialogue, et le modèle du contexte.

UsiXML est fréquemment utilisé dans les approches basées sur un modèle. Pour d'amples informations sur ce langage, le lecteur peut se référer au chapitre 2, §2.2.6, où nous détaillerons encore ce langage ainsi que son utilisation dans le cadre de l'Ingénierie Dirigée par les Modèles.

UIML

Le User Interface Markup Language (UIML) (Abrams *et al.*, 1999; Helms *et al.*, 2009) est un langage déclaratif permettant de décrire les interfaces utilisateur. L'objectif derrière la proposition de UIML est de fournir un langage indépendant de toute plate-forme et de toute modalité d'interaction, capable d'établir la correspondance avec d'autres langages de programmation, d'autres LDIU, ainsi que d'autres technologies

UIML est considéré comme étant un métalangage. En effet, ce langage propose un ensemble de balises prédéfinies d'une manière générique et qui sont indépendantes de toute modalité d'interaction, de toute plate-forme et de tout langage cible auquel UIML sera mappé. Pour pouvoir utiliser UIML convenablement, il faut ajouter un vocabulaire qui est défini comme étant un ensemble de composants d'interface, ainsi que leurs propriétés, leurs opérations et leurs évènements. Les vocabulaires définis peuvent avoir différents niveaux d'abstraction, allant des vocabulaires très abstraits aux vocabulaires très proches de la plateforme cible.

UIML sépare les éléments d'une interface utilisateur. Cette séparation identifie les parties suivantes : `interface` et `peers`. `Interface` représente la description de l'IHM à travers quatre parties :
- `structure`, qui représente l'organisation et les hiérarchies de toutes les parties composant l'IHM ;
- `content` qui décrit les données de l'application qui seront affichées ;
- `behavior` qui représente le comportement de l'application au moment de l'interaction avec l'utilisateur ; et
- `style` qui définit toutes les propriétés spécifiques pour chaque élément de l'IHM.

La partie `peers` lie les éléments génériques de l'IHM, à une plate-forme spécifique en utilisant les sous-parties :
- `présentation` qui contient des correspondances entre les parties, les évènements ainsi que les propriétés avec le vocabulaire cible ; et
- `logic` qui permet de faire le lien entre les méthodes externes et celles invoquées dans la partie `behavior`.

La séparation de ces aspects facilite à la fois la réutilisation des définitions d'interfaces et la cohérence entre les différentes plateformes. L'interface n'est décrite qu'une seule fois en utilisant un modèle qui fait une abstraction du type de logiciel et du matériel et qui

sera utilisé comme support final. De même, la définition de l'interconnexion de l'interface avec d'autres composants externes est faite une seule fois. La description de l'interface est ensuite rendue, conformément à ce qui est défini dans la partie présentation, et communique avec la logique d'application via la partie `logic`. UIML fournit la possibilité aux différentes parties de l'interface, d'être regroupées dans des modèles (nommé `template`). Les modèles peuvent ensuite être réutilisés dans la conception d'autres interfaces.

Pour obtenir une interface finale, il existe deux types du moteur de rendu, le premier type interprète le code UIML directement sur la machine cliente et génère l'interface et le deuxième type des moteurs traduit UIML vers un autre langage de programmation classique (Java, C#, HTML, etc.).

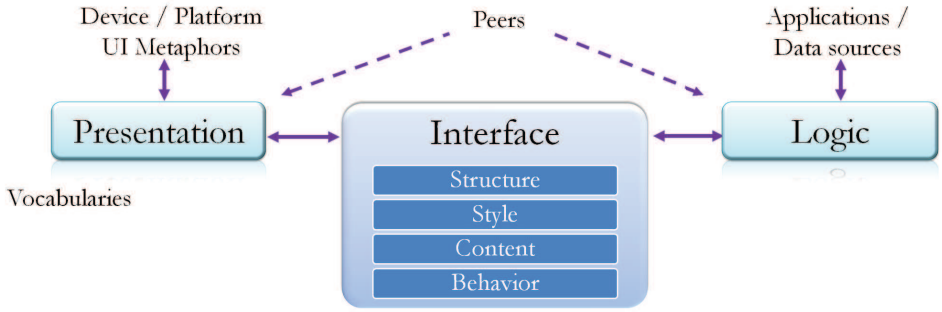


FIGURE 1.10 – L'architecture du UIML

1.6.3 Récapitulatif et synthèse

Dans les sous-sections précédentes, nous avons passé en revue certain nombre de Langages de Description des Interfaces Utilisateurs. Dans cette section, nous proposons une vue d'ensemble permettant une comparaison de ces langages citées précédemment. Afin de pouvoir spécifier et caractériser chaque langage, nous devons choisir les critères sur lesquels se base cette comparaison. Certains de ces critères ont été inspirés des travaux de Souchon et Vanderdonckt (2003), Trewin *et al.* (2004) et García *et al.* (2009), d'autres ont été identifiés en fonction des besoins et des exigences de notre problématique :

- **Modèles décrits :** ce critère décrit les aspects et les modèles que le langage permet de modéliser. Le *modèle de tâche* est une description de la tâche à accomplir par l'utilisateur, le *modèle de domaine* est une description des objets que l'utilisateur manipule, accède ou visualise à travers les interfaces, le *modèle de présentation* contient la représentation statique de l'interface utilisateur et le *modèle du dialogue* détient l'aspect conversationnel de l'interface utilisateur.
- **Méthodologie suivie :** il existe plusieurs approches pour modéliser et caractériser une interface sensible au contexte :
  - Spécification d'une description d'interface utilisateur pour chacun des différents contextes d'utilisation.

  - Spécification d'une description générique (ou abstraite) valable pour tous les contextes d'utilisation. Cette description d'interface utilisateur générique est ensuite affinée pour répondre aux exigences des différents contextes d'utilisation.

- **Prise en compte du contexte :** ce critère vise à indiquer la dimension du contexte pour laquelle le langage a été conçu (mono / multi-plateforme, mono / multi-utilisateur ou mono / multi-environnement).
- **Outils :** certains langages sont supportés par un outil qui aide le concepteur ou bien qui permet la transformation de ce langage vers d'autre langages ou d'autres plateformes spécifiques.
- **Langages supportés :** permet d'indiquer les langages de programmation auquel le LDIU peut être traduit.
- **Accès aux donnés externes :** ce critère permet de préciser la manière avec laquelle le langage permet d'accéder aux données. En effet, il existe deux manières pour le faire :
  - Le langage exige la connaissance préalable de la structure des données auxquelles il doit accéder.
  - Le langage n'exige pas la connaissance de la structure des donnés auxquelles il doit accéder et il permet l'accès à différentes sources de données externes indépendamment de leurs structures.
- **Suivi du flux d'information :** utile pour savoir si le langage permet de traduire la dynamique du passage du flux d'information décrite dans un modèle de tâche.

TABLE 1.4 – Tableau comparatif des Langages de Description des Interfaces Utilisateurs

	Modèles	**Méthodologie**	**Contexte**	**Langages supportés**	**Outils**	**Accès aux Données**		**Flux d'information**
						Données structurées	Données externes	
XIML	- Présentation - Domaine - Dialogue - Tâches	Description générique de l'interface	- Multi-plateformes	- HTML - WML - Java	Éditeur de code XIML	Oui	Non	Non
XForms	- Présentation - Domaine	Description spécifique pour chaque contexte	- Multi-utilisateurs	- HTML - XHTML - WML	Éditeur de code XForms	Oui	Non	Non
AUIML	- Présentation - Dialogue	Description générique de l'interface	- Multi-plateformes	- HTML - DHTML - Java Swing - PalmOS - WML	- AUIML Visual Builder - Plug-in Eclipse : Assistant et Générateur	Non	Non	Non
AAIML	- Présentation - Dialogue	Description générique de l'interface	- Multi-plateformes - Multi-utilisateurs	N'est pas définit	N'est pas définit	Non	Non	Non
UsiXML	- Présentation - Domaine - Dialogue - Tâches	Description générique de l'interface	- Multi-plate-formes, - Multi-utilisateurs - Multi-environnements	- Flash - VRML - WML - XHTML - C++ - Java - Java3D - VoiceXML	- SketchiXML - GrafiXML - FlashiXML - QtkXML - IdealXML - Transformi-XML	Oui	Non	Non
UIML	- Présentation - Domaine - Dialogue - Tâches	Description générique de l'interface	- Multi-plateformes	- HTML - Java - C++ - C# - VoiceXML - QT - CORBA - WML	- LiquidApps - UIML.net - VoiceXML renderer - WML renderer - VB2UMIL	Oui	Oui	Oui

De cette comparaison, il est possible de déduire cinq points :

- Les langages présentés auparavant partagent certaines caractéristiques. En effet, tous ces langages abstraits ont les architectures claires qui séparent les différents composants fonctionnels d'une interface. Cependant, l'ensemble des modèles constituant chaque architecture diffère d'une approche à une autre. UIML, UsiXML et XIML décrivent les aspects Présentation, Domaine, Dialogue et Tâches tandis que AUIML et AAIML se sont intéressés à la définition de la Présentation et du Dialogue. Quand à XForms, il s'est contenté de décrire la Présentation et le Domaine.

- UIML, UsiXML, AUIML, XIML et AAIML sont tous destinés à couvrir un large éventail de plateformes cibles. XForms parait plus restreint sur ce côté et s'intéresse plus à décrire l'utilisateur que la plateforme. Le choix de la dimension du contexte à cibler est fortement lié à la méthodologie adoptée par le langage. En effet, dans le cas de XForms, la spécification de l'interface s'effectue par rapport à un utilisateur bien déterminé, alors que pour le reste des langages présentés, la description de l'interface est effectué d'une manière générique, indépendamment de tout élément du contexte.

- Bien que la majorité des langages présentés s'intéressent au modèle du domaine, nous nous apercevons que l'accès à des ressources externes des données n'est fourni que par UIML. En effet, certains langages permettant la manipulation des données (XIML, XForms et UsiXML) exigent la connaissance au préalable de la structure de ces données. Par contre, UIML permet un accès à n'importe quel type de données en se basant sur l'invocation des méthodes externes, via la partie `logic`, dont la mission est d'extraire les données pour qu'elles soient exploitées, au moment du rendu de l'interface finale.

- La traduction de la dynamique du passage du flux d'informations entre les tâches ne peut être effectuée qu'a travers le langage UIML. En effet, la notion de variable introduite dans la version 4.0 de UIML [11] peut être exploitable pour ce fait. Les autres langages présentés s'intéressent uniquement à décrire les interactions entre l'utilisateur et le système.

- Les outils qui accompagnent les langages abstraits jouent un rôle essentiel, étant donné que certains d'entre eux permettent de couvrir le processus de conception tandis que le rôle des autres peut s'étendre pour générer l'interface finale. Les langages les plus matures au niveau outillage sont UsiXML et UIML, vu qu'ils proposent des éditeurs du code, des outils de manipulation des différents modèles du langage ainsi que les moteurs du rendu permettant de générer des interfaces finales. Dans un rang moins important, se présente AUIML avec son *plugin* Eclipse jouant le rôle d'assistant, ainsi qu'un générateur d'interface. Le reste des langages (XIML et XForms) se contentent de proposer uniquement des éditeurs de code.

11. http ://docs.oasis-open.org/uiml/ns/uiml4.0

Conclusion

Nous avons présenté dans ce premier chapitre une rapide introduction à l'Ingénierie dirigée par les modèles. Cette approche que nous adoptons pour nos travaux s'inscrivant dans le cadre de cette thèse étant donné sa capacité à accélérer les développements et de faciliter leur réutilisation. Nous avons également introduit une des notions clés de notre travail à savoir la personnalisation. Dans cette introduction, nous avons énuméré les différentes dimensions sur lesquelles s'applique cette personnalisation et nous nous sommes intéressés particulièrement au contenu et aux différents facteurs qui l'influencent. Nous avons mis en exergue le contexte d'utilisation, un des principaux facteurs révélés qui agissent sur la personnalisation du contenu.

Cet intérêt au contenu nous a conduit à introduire les ontologies, une des meilleures méthodes permettant de le présenter sémantiquement. En effet, les ontologies permettent de représenter formellement les connaissances et d'en décrire le raisonnement. Les deux dernières sections sont consacrées respectivement à la modélisation des tâches et aux langages de description des interfaces utilisateurs. Dans chaque partie, nous avons énuméré les formalismes permettant respectivement de modéliser les taches et de décrire les IHM et nous avons mené également une étude comparative entre les modèles de chaque spécialité.

Pour achever cette étude bibliographique, nous présenterons dans le chapitre suivant, un état de l'art sur les approches basées sur les modèles visant la génération des IHM sensibles au contexte.

Chapitre 2

Approches à base de modèles pour l'adaptation des IHM

Introduction

L'objectif initial des approches basées sur les modèles, nommées également *approches déclaratives*, était de permettre aux personnes qui ne sont pas nécessairement des informaticiens, de pouvoir concevoir et produire des IHM d'une manière automatique ou semi-automatique. En effet, la plupart des systèmes proposent des processus guidés par le concepteur plutôt que d'utiliser une approche entièrement automatique. Les approches déclaratives sont apparues à la fin des années 80 et au début des années 90, dans le but de simplifier le processus de création et de maintenance des IHM ainsi que de fournir un moyen permettant aux applications de fonctionner sur différentes plates-formes en partant d'un seul ensemble de modèles. Les premiers systèmes à base de modèles ont présenté plusieurs inconvénients. Plus particulièrement, la création des modèles nécessaires pour générer une interface a été un processus très abstrait et long à mettre en place. Souvent, le temps nécessaire pour créer les modèles dépassait le temps nécessaire pour programmer manuellement une IHM. Enfin, la génération automatique de l'interface utilisateur a été une tâche très difficile et aboutissait souvent à des interfaces de mauvaise qualité (Myers *et al.*, 2000). D'après (Gajos, 2008), dernièrement, avec l'amélioration des technologies utilisées, ce processus de conception s'est accéléré et les approches basées sur les modèles étaient employées pour la génération des systèmes interactifs sensibles au contexte.

Dans ce chapitre, nous nous intéressons uniquement à cette dernière catégorie d'approches et nous présentons un panel, non exhaustif, des principales approches basées sur les modèles visant l'adaptation ou la personnalisation des IHM, en considérant le contexte d'utilisation. Pour plus de détails sur les approches à base de modèles qui ne se focalisent pas sur l'adaptation des IHM (ex : (Szekely *et al.*, 1995; Barthet et Tarby, 1996; Bodart *et al.*, 1996)), le lecteur peut se référer aux thèses de (Tabary, 2001) et (Thevenin, 2001).

2.1 Critères de comparaison des approches

Afin de bien étudier les travaux que nous allons détailler dans ce chapitre et pouvoir situer notre travail parmi ces approches, nous proposons de choisir rigoureusement les critères sur lesquels se base cette étude. En effet, (Vanderdonckt *et al.*, 2005) ont proposé un espace de conception pour la prise en compte du contexte (cf. Figure 2.1). Cet espace est composé de deux parties : la partie supérieure indique le type de variation du contexte pouvant inciter l'adaptation de l'IHM et la partie inférieure représente les éléments impliqués dans l'adaptation. (Thevenin, 2001), à partir d'une collection de plusieurs taxonomies présentant des espaces d'adaptation des IHM, a suggéré une classification sous forme de fleur décrivant les outils de génération des IHM multi-cibles (cf. Figure 2.2). Elle comprend cinq axes : la nature de la cible couverte par l'adaptation, le type de l'IHM, les composants adaptés, les acteurs de l'adaptation et l'intervention de l'outil dans le processus de conception. En se basant sur les travaux précédemment présentés ainsi que sur les problématiques que traite notre approche, nous avons identifié les critères sur lesquels se base notre étude

comparative. Ces critères ont été répertoriés en trois volets majeurs : les critères orientés *modélisation*, les critères orientés *personnalisation* et les critères orientés *implémentation.*

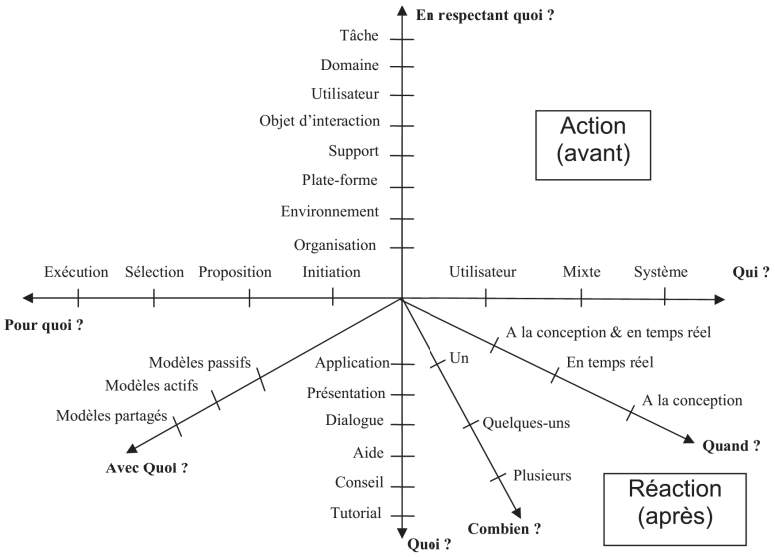


FIGURE 2.1 – Espace de conception pour la prise en compte du contexte (traduit de (Vanderdonckt *et al.*, 2005))

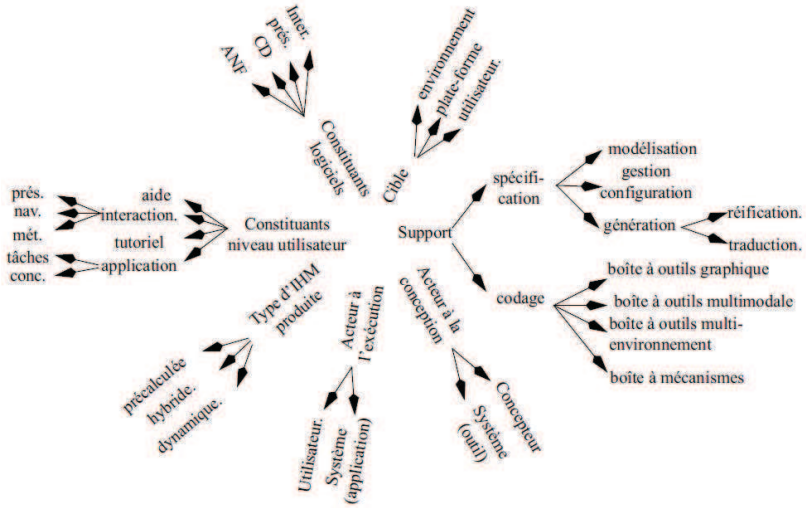


FIGURE 2.2 – La fleur des outils pour IHM multicibles (Thevenin, 2001)

2.1.1 Critères orientés modélisation

Ces critères, permettant de décrire et de caractériser l'ensemble des modèles d'une approche spécifique, sont les suivants :
- *Les modèles de l'approche* : représentent l'ensemble des modèles utilisés pour la mise en œuvre de l'approche indépendamment de leurs niveaux d'abstraction.
- *La conformité au MDA* : indique si l'approche est conforme ou pas à l'architecture MDA.
- *La transformation des modèles* : décrit la manière avec laquelle s'effectue la transformation de l'ensemble des modèles de l'approche.

2.1.2 Critères orientés personnalisation

Ces critères servent à évaluer la réussite de l'approche en termes de mise en œuvre de la personnalisation. Nous avons identifié cinq critères se résumant ainsi :
- *La cible d'adaptation* : permet de répondre à la question "Quoi ?" posée par (Vanderdonckt *et al.*, 2005) pour désigner l'élément sur lequel porte l'adaptation. L'ensemble de ces éléments a été présenté dans le chapitre1, §1.2.2.
- *Les éléments du contexte* : utiles pour répondre à la question "En respectant quoi ?" posée par (Vanderdonckt *et al.*, 2005) afin de spécifier l'ensemble des éléments du contexte à prendre en compte pendant l'adaptation de l'interface. (Thevenin, 2001) a qualifié ce critère par le terme "*cible*" qui se définit par le triplet ≺Utilisateur - Plateforme - Environnement≻.
- *La modélisation du contexte* : indique si l'approche propose un modèle de contexte et quel est le niveau d'abstraction de cette modélisation.
- *La prise en compte du contexte* : accorde la possibilité de savoir la manière avec laquelle le contexte à été pris en compte pour la mise en œuvre de l'approche.
- *Le moment de l'adaptation* : sert à répondre à la question "Quand ?" posée par (Vanderdonckt *et al.*, 2005) afin de connaitre le moment de l'adaptation de l'IHM. Cette adaptation peut être réalisée *pendant la conception* de l'IHM (*Design time*), *en temps réel* (*Runtime*) pendant l'exécution de l'application ou bien *les deux ensemble* si l'adaptation s'effectue pendant les deux phases.

2.1.3 Critères orientés implémentation

Ces critères permettent de mesurer la maturité de l'approche, au niveau technique, en termes d'outillage proposé et en termes d'implémentation des modèles proposés. Dans le cadre de notre étude, nous avons proposé les critères suivants :
- *Langage utilisé* : représente le langage de spécification adopté par l'approche pour implémenter ses modèles.

– *Outillage* : décrit l'ensemble d'outils proposés par chaque approche. Nous prenons en compte l'ensemble des outils proposés de la phase de conception jusqu'à la phase de génération de l'IHM.

2.2 Présentation des approches étudiées

Dans cette section, nous allons détailler l'ensemble des approches basées sur les modèles visant l'adaptation des IHM au contexte. Chaque approche sera étudiée et suivie d'un tableau récapitulatif représentant une synthèse de l'approche en se référent aux différents critères annoncés. La présentation des approches sera effectuée dans l'ordre croissant de leur apparition.

2.2.1 CAMELEON

(Calvary *et al.*, 2003) ont proposé dans CAMELEON un cadre de référence, pour les approches déclaratives qui s'intéressent à la génération des IHM plastiques sensibles au contexte. Pour les auteurs, une interface sensible au contexte présente des capacités d'adaptation face au changement du contexte constitué par le triplet ≺Utilisateur - Plateforme - Environnement≻.

Pendant le processus de génération d'interface, le cadre de référence distingue 4 niveaux d'abstraction :
– Niveau Tâches et Concepts qui décrit la sémantique de l'application du point de vue de l'utilisateur. Les modèles utilisés dans ce niveau sont indépendants de toutes plateformes.
– Niveau Interface Abstraite (IUA) qui modélise l'interface d'une manière abstraite et précise le dialogue entre l'utilisateur et l'interface. Dès ce niveau abstrait, le concepteur peut décider sur le mode d'interaction (graphique, vocale, etc).
– Niveau Interface Concrète (IUC) permettant de spécifier l'IHM en termes d'intéracteurs concrets.
– Niveau Interface Finale (IUF) qui modélise l'IHM finale dans une plateforme particulière.

Les modèles proposés dans CAMELEON sont répartis en 3 catégories : ***les modèles ontologiques*** (modèle de domaine – modèle de contexte – modèle d'adaptation) qui forment la base des autres modèles et qui peuvent être instanciés, soit en ***modèles archétypaux*** utilisés pendant le processus de la conception, soit en ***modèle observés*** permettant l'adaptation de l'interface pendant l'exécution. Le processus est défini comme étant une combinaison de réifications verticales et de translations horizontales : La réification permet de passer, dans l'architecture, d'un niveau abstrait vers un autre moins abstrait tandis que la translation permet de passer d'un contexte d'utilisation vers un autre tout en gardant le même niveau d'abstraction.

CAMELEON a connu plusieurs versions et plusieurs améliorations qui se sont étalées de l'année 2000 jusqu'à l'année 2003. Au début, (Calvary *et al.*, 2000, 2001) ont proposé l'allure globale de leur approche en termes de modèles et de nivaux d'abstraction. Ensuite, dans une version révisée du framework (Calvary *et al.*, 2002), les auteurs ont introduit la prise en compte de l'adaptation de l'interface pendant l'exécution (le Runtime). Dans la version suivante (Calvary *et al.*, 2003), le modèle de contexte d'utilisation a été étendu en rajoutant la dimension *"Utilisateur"* aux autres dimensions du contexte, à savoir l'environnement et la plateforme. L'architecture finale de CAMELEON est présentée dans la Figure 2.3.

Étant donné que CAMELEON présente un cadre de référence, les modèles proposés n'étaient pas développés et détaillés. Par conséquent, nous ne présentons pas de tableau de synthèse.

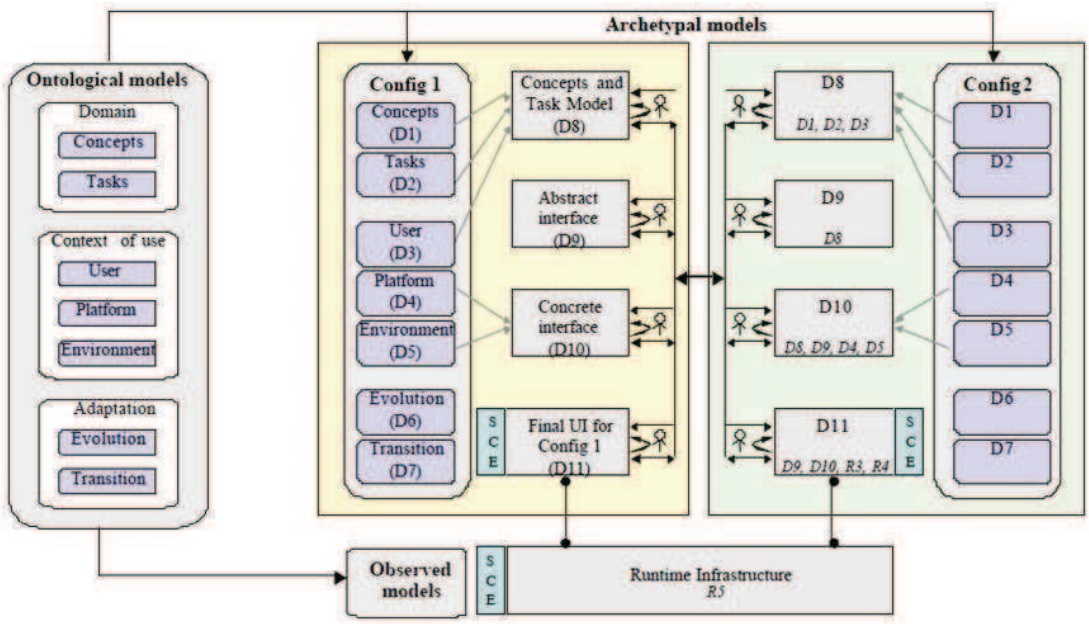


FIGURE 2.3 – Le framework Caméleon (Calvary *et al.*, 2003)

2.2.2 ArtStudio

ArtStudio (Adaptation par Réification et Traduction Studio) (Thevenin, 2001) est un environnement logiciel pour le développement des interfaces plastiques multi-cible.

Il a été élaboré en se basant sur le cadre de référence CAMELEON dans une de ces premières versions (Calvary *et al.*, 2001). En effet, pendant le processus de modélisation, ArtStudio soutient le processus de réification en suivant les quatre niveaux d'abstraction fournis par CAMELEON. Pour cette raison, plusieurs modèles ont été proposés dans

l'approche, à savoir : le modèle de concepts (en UML) ; le modèle des tâches (inspiré de CTTe) ; le modèle de la plateforme ; le modèle des interacteurs ; le modèle abstrait de l'interface et, finalement, le modèle concret de l'interface.

Comme proposé par CAMELEON, le processus de développement est une combinaison de réification verticale et de translation horizontale. Une réification verticale est appliquée, entre les différents niveau d'abstraction, pour une plateforme cible particulière, tandis que la translation est utilisée pour exprimer le passage entre différentes cibles.

Pendant le passage de l'IUA à l'IUC, chaque élément abstrait appelé Unité de Présentation (UP) est associé à un élément concret. La correspondance se fait en fonction des éléments suivants :
– Le sens de l'information (Entrée/Sortie)
– Le type de l'information (Entier, Chaine de caractères...)
Après avoir été générée, l'IUA ne peut pas être modifiée alors qu'après la génération de l'IUC, cette dernière peut être changée. La correspondance entre intéracteurs et éléments finals de l'interface se fait en se basant sur une fonction qui calcule le coût de l'exécution (en prenant en compte les caractéristiques physiques de l'interface finale). L'interface finale obtenue (IUF) est exprimée dans un code source cible.

ArtStudio fait partie des premiers environnements respectant le cadre de référence CAMELEON. Cependant, l'approche ArtStudio est limitée à des plates-formes graphiques Java et des pages Web. En plus, cette approche ne s'intéresse qu'à l'adaptation de la présentation de l'interface par rapport à la plateforme, et ce, pendant le design time. Le Tableau 2.1 résume l'approche ArtStudio.

Selon (Sottet, 2008), le point fort d'ArtStudio réside dans la génération d'IHM multi-cibles étant donné que cette approche embarque un ensemble de règles pour choisir l'interface la plus appropriée à la plate-forme cible. (Sottet, 2008) affirme aussi que la faiblesse de ce système réside dans l'indéterminisme du choix des heuristiques de génération. En plus, l'implémentation de l'outil ARTStudio est incomplète selon Sendin *et al.* (2008).

TABLE 2.1 – ArtStudio en synthèse

Modélisation	Modèles de l'approche	- Modèle des tâches - Modèle de domaine - IU Abstraite - IU Concrète - IU Finale - Intéracteurs
	Conformité au MDA	Non
	Transformation des modèles	Ensemble de règles pour choisir l'interface la plus appropriée à la plate-forme cible.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu (plasticité)
	Moment d'adaptation	Design time
	Éléments du contexte	≺Plateforme ≻
	Modélisation du contexte	Modélisation de la plateforme en UML
	Prise en compte du contexte	Utilisation des caractéristiques de l'interface dans les règles de transformations.
Implémentation	Langage de spécification	XML
	Outils associés	ArtStudio

2.2.3 Dygimes

Dygimes (Coninx *et al.*, 2003; Luyten *et al.*, 2003) présente un framework permettant de générer automatiquement des interfaces homme machine pour le domaine mobile, les systèmes embarqués et qui peut être étendu même pour les interfaces distribuées.

Ce framework permet une séparation de l'interface utilisateur du code d'application. Par conséquent, en utilisant les fonctionnalités standard, l'interface qui en résulte peut être utilisée sur de nombreux plateformes, permettant la réutilisation souple des modèles existants.

Dygimes a utilisé le concept de la séparation des couches faisant une distinction claire entre les modèles de mise en œuvre de chaque couche. Le concepteur peut commencer par un modèle des tâches en utilisant la notation CTT (Paterno, 1999). Ce modèle peut être enrichi par des blocs de construction contenant l'ensemble des composants de l'interface et qui sont fournis sous format Seescoa XML. Après l'obtention d'un modèle des tâches annoté, ce dernier subit une transformation pour en déduire un modèle de dialogue permettant de décrire les transitions entre les différents états que peut prendre l'interface.

Le modèle de dialogue, qui décrit le comportement du système, est implémenté en utilisant la notation Enabled Task Sets (ETSs). Ensuite, les éléments abstraits de l'interface seront obtenus en traduisant le code Seescoa XML. Pendant le passage du niveau abstrait au niveau concret, plusieurs facteurs peuvent être intégrés (par exemple : les contraintes de présentation, le profil de la plateforme ainsi que les éléments concrets de l'interfaces qui sont disponibles). Le code généré sera ensuite interprété pour produire l'interface finale.

Dans leur architecture, les auteurs n'intègrent pas la prise en compte du contexte comme défini dans CAMELEON (Calvary *et al.*, 2003), mais ils se contentent de la prise en compte que de la plateforme afin d'adapter la présentation de l'interface. Tel que c'est le cas pour CAMELEON, Dygimes présente un cadre de référence. Par conséquent, nous ne présentons pas de tableau de synthèse étant donné que les modèles proposés n'étaient pas développés.

2.2.4 TERESA

TERESA (Transformation Environment for InteRactivE Systems representAtions) (Mori *et al.*, 2004; Berti *et al.*, 2004) est un environnement pour la génération semi-automatique d'interfaces multi-plateforme et multi-modale (graphique et vocale). Le principe suivi dans cette approche était « *One Model, Many Interfaces* ». Pour pouvoir mettre en œuvre ce principe, les auteurs ont adopté le cadre de référence CAMELEON (Calvary *et al.*, 2003) basé sur l'utilisation d'un seul modèle de départ pour en dériver plusieurs interfaces cibles.

La modélisation commence par l'établissement d'un modèle des tâches qui s'appuie sur la notation CTT (Paterno, 2000) (cf. chapitre1, §1.5.3). Ce modèle peut être annoté et enrichi. Ainsi pour chaque tâche, le concepteur peut spécifier le concept de domaine relié et manipulé par cette tâche ainsi que la plateforme cible sur laquelle la tâche peut être réalisée. TERESA s'appuie sur l'outil CTTE qui permet de concevoir le modèle des tâches. Pour valider le modèle effectué, une simulation du scénario modélisé est offerte par l'outil afin de vérifier la cohérence et le bon déroulement des tâches. Après validation du modèle des tâches, une spécification abstraite de l'interface (IUA) peut être obtenue en transformant celui ci. Il s'agit d'une opération semi-automatique dans laquelle l'outil analyse les modèles des tâches (tâches annotées et relations temporelles). Cette analyse inclut l'utilisateur dans le choix des heuristiques à effectuer pendant le passage vers l'AUI.

Avant de générer l'interface finale, le concepteur peut obtenir la spécification concrète de l'interface (IUC) qui sera déduite du modèle précédent et qui sera dépendante de la plateforme finale. TERESA permet au concepteur d'intervenir et d'effectuer des changements au niveau de l'IUC. C'est seulement après la transformation du CUI que le concepteur peut générer son interface finale adaptée à une plateforme cible bien déterminée.. Pendant toutes les phases de conception, TERESA s'appuie sur la notation XML, afin de sérialiser ses modèles. (cf. Tableau 2.2).

Cette approche basée sur les modèles permet uniquement l'adaptation de l'interface (contenant) par rapport à la plateforme, et ce, pendant le design time.

Table 2.2 – TERESA en synthèse

Modélisation	Modèles de l'approche	- IU Abstraite - IU Concrète - IU Finale - Modèle des tâches - Modèle de domaine
	Conformité au MDA	Non
	Transformation des modèles	Génération assisté par le concepteur à travers le choix de la plateforme finale et le choix des heuristiques de transformation.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu
	Moment d'adaptation	Design time & Runtime
	Éléments du contexte	≺Plateforme ≻
	Modélisation du contexte	Pas de modèle fourni
	Prise en compte du contexte	Prise en compte de la plateforme par annotation du modèle des tâches.
Implémentation	Langage de spécification	- CTT (Concurrent Task Trees) - XML - TERESA XML
	Outils associés	- TERESA - CTTE (Concurrent Task Trees Environement)

2.2.5 SUPPLE

SUPPLE (Gajos et Weld, 2004; Gajos *et al.*, 2010) est une approche et un outil qui traitent l'adaptation des interfaces comme étant un problème d'optimisation contrairement aux autres approches de génération automatique d'interfaces basées généralement sur des règles de transformations.

Afin de générer des interfaces adaptées à l'utilisateur, ses préférences et ses capacités, les auteurs ont eu recours à un ensemble de modèles : modèle abstrait, modèle de la plateforme et modèle des traces des évènements créés par un utilisateur.

Pour mettre en œuvre son approche, avant la génération de l'interface, l'outil SUPPLE teste la capacité de l'utilisateur à contrôler le système. Cette phase sert à susciter les préférences de l'utilisateur ainsi que ses capacités physiques. En même temps, le système enregistre les traces d'interaction de l'utilisateur en calculant le coût émis pour effectuer chaque tâche. En fait, sur la base du test effectué, la personnalisation consiste à fournir à l'utilisateur les fonctionnalités les plus fréquemment utilisées. SUPPLE fournit les modèles suivants :

– La spécification de l'interface : permet une description fonctionnelle de l'interface qui s'intéresse aux données manipulées (entiers, flottants, chaînes, les booléens, dates, heures, images, etc.) et non pas aux buts de l'utilisateur. La notation utilisée est légèrement différente du CTT. Chaque interface est constituée par un ensemble de couples contenant l'élément de l'interface et les contraintes subies.
– Le modèle de plateforme : permet de décrire l'interface de l'utilisateur à travers les éléments de l'interface qui sont valables, les contraintes qu'ils subissent ainsi que les fonctions calculant l'élément le plus convenable à l'utilisateur.
– Le modèle des traces de l'utilisateur : quand l'utilisateur demande l'affichage d'une interface sur une plateforme spécifique, SUPPLE cherche un compromis entre les contraintes de la plateforme et les traces de l'utilisateur afin de choisir les éléments de l'interface les plus appropriés à ses préférences (ceux qui nécessitent le moindre effort de la part de l'utilisateur pendant l'interaction).

Bien que cette approche intègre les préférences de l'utilisateur dans le processus d'adaptation de l'interface, elle ne prend pas en compte l'environnement de l'interaction pendant les transformations et s'intéresse uniquement à adapter le contenant et non pas le contenu (cf. Tableau 2.3).

Table 2.3 – SUPPLE en synthèse

Modélisation	Modèles de l'approche	- Modèle abstrait de l'interface - Modèle de l'utilisateur - Modèle de la plateforme
	Conformité au MDA	Non
	Transformation des modèles	Étant donné que les modèles sont sous forme de n-uplet, la transformation de ces modèles se traite comme étant un problème d'optimisation.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu
	Moment d'adaptation	Design time & Runtime
	Éléments du contexte	≺Plateforme - Environnement≻
	Modélisation du contexte	Modélisation sous forme de n-uplets
	Prise en compte du contexte	Effectuer un calcul entre les contraintes de l'interface et le modèle de l'utilisateur.
Implémentation	Langage de spécification	Notation qui ressemble à CTT (description fonctionnelle de l'interface se focalisant sur la nature des données manipulées)
	Outils associés	SUPPLE

2.2.6 UsiXML

En se basant sur le cadre de référence CAMELEON, les auteurs proposent un environnement pour la génération des interfaces multi-plateformes, multimodales et multi-langages et pour différents contextes d'utilisation, dans une approche de type MDA appelée UsiXML (User interface eXtensible Markup Language) (Limbourg *et al.*, 2005).

UsiXML fournit un haut niveau d'abstraction pour la conception des IHM, en utilisant un ensemble de modèles impliqués dans la conception des IHM et en fournissant un langage complet de modélisation des IHM basé sur XML, appelé le langage UsiXML (UsiXML, 2007). Dans cette approche, les auteurs ont fourni l'ensemble des modèles suivants :
– Modèle de domaine
– Modèle des tâches
– Modèle d'interface utilisateur abstrait (AUI)
– Modèle d'interface utilisateur concret (CUI)
– Modèle de contexte
– Modèle de transformation
– Modèle de mapping

(Vanderdonckt, 2005) considère que le niveau CIM contient le modèle des tâches ainsi que les concepts du domaine définis par un modèle de domaine. Le niveau PIM correspond à l'IUA définie au niveau de CAMELEON (Calvary *et al.*, 2003) et qui représente l'ensemble d'éléments permettant de décrire l'interface d'une manière abstraite. L'IUC est définie au niveau PSM afin de décrire l'interface d'une façon concrète à travers un ensemble d'éléments fournit par UsiXML. Pour le passage d'un niveau à un autre, UsiXML propose un modèle de transformation facilitant la transformation d'un modèle à un autre (De CIM vers PIM, de PIM vers PSM et de PSM vers l'interface finale et inversement). Les auteurs ont défini un modèle de mapping qui contient les correspondances entre les différents modèles de UsiXML et ils utilisent une technique de transformation de graphes pour effectuer la transformation entre ces modèles.

De toutes les approches déclaratives permettant de prendre en charge l'adaptation des IHM, UsiXML est considérée actuellement parmi les plus matures (Brossard, 2008; Samaan, 2006) en proposant une méthodologie, un langage complet de modélisation des IHM basé sur XML (UsiXML) ainsi qu'un nombre important d'outils permettant la mise en œuvre de cette approche. Le Tableau 2.4 résume UsiXML.

UsiXML explore plus la dimension plateforme que les autres dimensions, puisque son but principal est de soutenir l'adaptation de la présentation de l'IHM. En effet, le modèle de l'environnement prend en compte uniquement trois aspects (lumière, bruit et stress). Le modèle de l'utilisateur était limité dans les premières versions de UsiXML (UsiXML, 2007) à quelques attributs qui décrivent l'expérience de l'utilisateur avec la plateforme. Il vient d'être récemment modifié afin de prendre en compte plus d'aspects (Tesoriero et Vanderdonckt, 2010). Ce modèle d'utilisateur est défini dans un méta-niveau qui nécessite une instanciation spécifique pour chaque domaine d'application.

TABLE 2.4 – UsiXML en synthèse

Modélisation	Modèles de l'approche	- Modèle de domaine - Modèle des tâches - IU Abstraite - IU Concrète - Modèle de transformation - Modèled de contexte - Modèle de mapping
	Conformité au MDA	Oui
	Transformation des modèles	Utilise une technique de transformation de graphes pour effectuer la transformation des modèles.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu
	Moment d'adaptation	Design time
	Éléments du contexte	≺Utilisateur - Plateforme - Environnement≻
	Modélisation du contexte	Proposition d'un modèle de contexte
	Prise en compte du contexte	Prise en compte des caractéristiques de la plateforme cible, lors du passage du niveau CUI vers l'interface finale.
Implémentation	Langage de spécification	UsiXML
	Outils associés	SketchiXML, GrafiXML, FlashiXML, QtkXML, IdealXML, TransformiXML

2.2.7 Dynamo-Aid

DynaMO-AID (Clerckx *et al.*, 2005; Cuppens *et al.*, 2005) prévoit un processus de conception des IHM sensibles au contexte et multiplateforme ainsi qu'un outil de conception.

Cette approche se base sur le framework Dygimes. Dans ce sens, les auteurs ont amélioré son architecture afin de permettre l'adaptation des interfaces au contexte d'utilisation. Certains modèles ont été ajoutés et d'autres ont été modifiés. Dynamo-Aid se base sur les modèles suivants :

– Le modèle dynamique des tâches : Étant donné que les anciens modèles des tâches proposés dans Dygimes ne permettent pas la mise en œuvre de l'intégration du contexte, les auteurs de Dynamo-Aid ont proposé un autre modèle se basant sur la notation CTT. Ce modèle permet d'annoter les tâches et d'en définir des nœuds de décision. Ces nœuds permettent, en fonction du contexte, de choisir la sous branche du modèle des tâches à exécuter.

– Le modèle dynamique de dialogue : Il sert à décrire les transitions entre les différents états que peut prendre l'interface. Les auteurs ont gardé la notation State Transition Network (STN) afin d'implémenter ce modèle. Ce dernier est obtenu suite à une transformation du modèle des tâches.
– Modèle dynamique de l'environnement : Il permet de connaitre la manière avec laquelle il faut agir au moment d'un changement du contexte.
– Modèle dynamique d'application : Il fait référence à la notion de service par des tâches. L'intégration d'un service consiste à l'ajouter au niveau du modèle des tâches et lui associer une tâche décisionnelle. Le fonctionnement de l'application change à l'apparition ou la disparition d'un service. A partir du moment où un service devient présent, le modèle de dialogue et d'environnement doivent changer impérativement.
– Modèle de présentation : Chaque interface peut être vue pendant tout le processus de conception et le concepteur peut l'enrichir par d'autres éléments tels que les layouts et les éléments de navigation.

Table 2.5 – Dynamo-Aid en synthèse

Modélisation	Modèles de l'approche	- IU Abstraite - IU Concrète - Modèle dynamique des tâches - Modèle dynamique de dialogue - Modèle dynamique de l'environnement - Modèle dynamique d'application - Modèle de présentation
	Conformité au MDA	Non
	Transformation des modèles	Les heuristiques de transformations sont propres à l'outil et ne sont pas fournis. Durant la transformation, l'intervention humaine est possible.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu
	Moment d'adaptation	Design time & Runtime
	Éléments du contexte	$\prec$Plateforme - Environnement $\succ$
	Modélisation du contexte	Pas de modèle fourni
	Prise en compte du contexte	Prise en compte de la plateforme par annotation du modèle des tâches.
Implémentation	Langage de spécification	- CTT - XML
	Outils associés	Dynamo-Aid

Dynamo-Aid (Clerckx *et al.*, 2005; Cuppens *et al.*, 2005) présente non seulement une approche mais aussi un outil de génération automatique d'IHM sensibles au contexte d'utilisation. Cette approche tient compte de la plateforme et de l'environnement, mais elle omet l'utilisateur et ne s'intéresse qu'à l'adaptation de la présentation du contenant et non à celle du contenu (cf. Tableau 2.5).

2.2.8 PERCOMOM

Les travaux de recherche (Brossard *et al.*, 2007; Brossard, 2008; Brossard *et al.*, 2011), qui ont conduit à la création de la méthode PERCOMOM font partie de la réalisation d'une plateforme de développement pour la génération d'un système d'information interactif personnalisé dans le domaine du transport collectif. Pour cette raison, une architecture globale, fondée sur une approche MDA, a été définie.

(Brossard, 2008) a défini 14 modèles répartis en modèles sociaux, modèles environnementaux, modèles comportementaux et modèles d'interaction. Parmi les modèles les plus développés de l'approche, nous citons *le modèle de processus métier* ainsi que le *modèle statique d'interaction*. Ces deux modèles constituent le point d'entrée pour la conception des applications étant donné qu'ils font partie du niveau CIM de l'architecture proposée. Le modèle de processus métier définit l'ensemble des tâches permettant d'atteindre un but métier. Pour sa mise en place les auteurs proposent l'adoption de la notation BPMN tout en l'enrichissant à fin d'y intégrer des éléments de personnalisation. Le modèle statique d'interface représente une abstraction des éléments constituant l'interface avec lesquels l'utilisateur final peut interagir.

Au niveau PIM, l'intégration et le développement d'un ensemble de services fonctionnels permettent la personnalisation du contenu. Ces services vont être utilisés aux niveau CIM pour annoter le modèle des processus métier.

Le niveau PSM de PERCOMOM a été divisé en deux sous-parties : la première représente un framework pour une famille d'IHM tandis que la deuxième représente un framework pour une IHM spécifique. Pour chaque niveau, il existe un moteur de règles offrant la possibilité d'adapter le comportement de chaque élément du framework en fonction d'un certain nombre de critères liés au contexte.

Pour intégrer le contexte depuis la phase de conception, l'approche propose l'utilisation des informations de contexte en tant que restrictions, annotant le modèle métier développé en BPMN. Ces restrictions seront marquées sur les arcs du modèle BPMN et seront considérées comme des conditions, pour passer d'une tâche à une autre.

La deuxième manière proposée dans PERCOMOM pour prendre en compte le contexte, dès les premiers stades de conception, est d'attacher aux éléments du modèle statique d'interaction, les services fonctionnels de personnalisation qui leur sont convenables.

Bien que PERCOMOM prenne en compte les différentes dimensions du contexte d'utilisation tel qu'il a été défini dans (Calvary *et al.*, 2003), aucune implémentation de ces

dimensions n'a été proposée dans l'approche. Le Tableau 2.6 synthétise l'approche PERCOMOM.

Table 2.6 – PERCOMOM en synthèse

Modélisation	Modèles de l'approche	14 modèles répartis en - Modèles sociaux - Modèles environnementaux - Modèles comportementaux - Modèles d'interactions
	Conformité au MDA	Oui
	Transformation des modèles	Comme la plupart des frameworks de l'approche n'ont pas été créés, les règles de transformation entre les différents niveaux de l'architecture ne sont pas implémentés.
Personnalisation	Cible d'adaptation	Adaptation du contenu
	Moment d'adaptation	Design time & Runtime
	Éléments du contexte	≺Utilisateur - Plateforme - Environnement≻
	Modélisation du contexte	Pas de modèle fourni
	Prise en compte du contexte	- Intégration des restrictions sur les éléments du contexte dans le modèle de processus métier - Utilisation des services de personnalisation du niveau PIM et leurs associations aux éléments du modèle statique d'interface.
Implémentation	Langage de spécification	XML
	Outils associés	Outil de modélisation du modèle métier et du modèle statique d'interaction.

2.2.9 Approche de (Sottet *et al.*, 2007)

Les auteurs proposent une approche de type IDM permettant la génération des IHM plastiques(Sottet *et al.*, 2007). Pour sa mise en œuvre, ils définissent un ensemble de principes à suivre : 1) Le système est un graphe de modèles qui sont interconnectés ; 2) Les transformations sont des modèles et peuvent eux même subir des transformations pendant le runtime ; 3) Le choix du Framework de mesure d'utilisabilité est libre ; 4) Le concepteur et l'utilisateur sont intégrés dans le processus de transformation des modèles.

Pour mettre en place leur approche, les auteurs ont adopté les modèles proposés par CAMELEON (Calvary *et al.*, 2003) et ont proposé une architecture permettant l'adaptation de l'interface pendant le runtime.

Le processus de cette adaptation est le suivant : Le manager des modèles contient un ensemble d'observateurs pour chaque modèle et dès qu'un changement se produit au niveau d'un modèle (arrivée d'un nouveau dispositif d'interaction sur le modèle de plateforme ou bien une interaction au niveau de l'interface finale...), le manager des modèles envoie une notification au moteur d'évolution ou au moteur de re-conception (dans le cas d'un changement au niveau du modèle du user). Le moteur d'évolution est constitué d'un "sélecteur de règles" et d'un manager de politiques. Les règles d'adaptation "adaptation rules", obéissent aux systèmes (Évènement - Condition – Action) : dès qu'un évènement stimule une règle, celle ci va se déclencher. Une règle est appliquée de deux manières : soit elle déclenche une transformation (qui sera exécutée par le moteur de transformation), soit elle appelle une autre règle d'adaptation. Ensuite, le moteur d'évolution identifie les transformations adéquates et les politiques à suivre pour réaliser l'adaptation. Ces transformations sélectionnées par le moteur d'évolution sont transmises au moteur de transformation pour effectuer la transformation. Enfin, l'interpréteur du IUC/IUF utilise la nouvelle IUC pour produire la nouvelle interface finale adaptée.

Table 2.7 – Approche de (Sottet *et al.*, 2007) en synthèse

Modélisation	Modèles de l'approche	- Modèle des tâches - IU Abstraite - IU Concrète - Modèle de transformation - Modèle de contexte
	Conformité au MDA	Non
	Transformation des modèles	Propose des règles de transformation en ATL dont les paramètres sont des éléments du contexte.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu
	Moment d'adaptation	Design time & Runtime
	Éléments du contexte	≺Utilisateur - Plateforme - Environnement≻
	Modélisation du contexte	Pas de modèle fourni
	Prise en compte du contexte	Le contexte est intégré dans les règles d'adaptation
Implémentation	Langage de spécification	
	Outils associés	ATL

Bien que l'approche proposée permette l'adaptation de l'interface en temps réel en tenant compte du contexte d'utilisation et en préservant son utilisabilité, les règles d'adaptation

restent non génériques étant donné qu'elles sont consacrées à des situations prédéfinies du contexte connues à l'avance. En cas d'ajout d'un nouvel élément du contexte, il faut ajouter aussi les règles de transformation qui lui sont spécifiques. Le Tableau 2.7 résume l'approche ArtStudio.

2.2.10 Approche de (Hachani *et al.*, 2009)

Les auteurs de cette approche (Hachani *et al.*, 2009) proposent une méthode générique pour l'adaptation des IHM sensibles au contexte. Ils suggèrent l'adaptation du modèle des tâches au contexte en développant des règles de transformation génériques et réutilisables appropriées à tous les contextes d'utilisation.

Pour mettre en œuvre leur méthodologie, les auteurs ont étendu l'approche de Sottet (Sottet *et al.*, 2007). Ils proposent une extension du modèle des tâches et définissent un métamodèle des tâches adaptable au contexte. De même, ils ont généralisé le modèle de contexte en proposant un métamodèle générique de contexte. Le métamodèle des tâches et celui du contexte sont compatibles avec les règles génériques de transformation qu'ils ont définies. (cf. Tableau 2.8).

Pendant la phase de conception, l'idée est de partir d'un modèle des tâches et d'y identifier les éléments fixes et variables dans le système. De telles tâches sont appelées « *des points de variation* ». Étant donné que le modèle des tâches proposé est adaptable au contexte, il faut annoter ce modèle (les tâches variables) pendant le design time, par des éléments du contexte en mentionnant le paramètre (élément du contexte) à prendre en compte dans la tâche. Ces éléments vont servir pour se décider pendant la phase de transformation du modèle quelles sont les tâches à conserver et celles à éliminer. Le modèle des tâches de départ «adaptable» va être réduit vers un autre qui est spécifique à un contexte bien déterminé. Ce contexte est défini par un profil de contexte qui contient les paramètres du contexte. Le modèle cible est obtenu en appliquant les règles de transformation génériques développées en langage ATL sur le modèle des tâches annoté.

Bien que dans l'approche proposée, les auteurs effectuent une adaptation de l'interface à la langue de l'utilisateur et à la taille de l'écran, ces deux aspects font partie de la dimension "adaptation de la présentation" et non pas celle du contenu.

TABLE 2.8 – Approche de (Hachani *et al.*, 2009) en synthèse

Modélisation	Modèles de l'approche	- Modèle des tâches adaptable au contexte - Métamodèle de contexte
	Conformité au MDA	Non
	Transformation des modèles	Définition des règles génériques et réutilisables appropriées à plusieurs contextes d'utilisation.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu
	Moment d'adaptation	Design time & Runtime
	Éléments du contexte	≺Utilisateur - Plateforme - Environnement≻
	Modélisation du contexte	Métamodèle de contexte
	Prise en compte du contexte	Prise en compte des éléments du contexte par annotation du modèle des tâches
Implémentation	Langage de spécification	
	Outils associés	ATL

2.2.11 Approche de (Bouchelligua *et al.*, 2010)

Les auteurs ont proposé une approche basée sur IDM permettant la génération des IHM plastiques (Bouchelligua *et al.*, 2010). L'adaptation de l'interface générée se fait en respectant les différents éléments du contexte d'utilisation tels qu'ils ont été spécifiés dans (Calvary *et al.*, 2003) et en se basant sur la transformation paramétrée (Vale et Hammoudi, 2008).

Comme cette approche se base sur le cadre de référence CAMELEON (Calvary *et al.*, 2003), les auteurs ont eu recours à certains modèles découlant de ce Framework, à savoir le modèle abstrait d'interface et le modèle concret d'interface.

En partant d'un modèle abstrait d'interface, la première transformation se base sur le modèle de l'utilisateur pour produire le modèle concret d'interface du premier niveau. Ce modèle produit subit lui aussi une transformation en tenant compte des caractéristiques de la plateforme. Et finalement, ce dernier modèle supporte une dernière transformation intègrant un modèle de l'environnement pour produire l'interface finale.

Dans cette approche, les auteurs fournissent les méta-modèles des différentes dimensions du contexte, utilisées pour adapter l'interface. L'adaptation se fait pendant le design time et elle ne considère que l'aspect présentation du contenu (contenant) et non pas le contenu. Le Tableau 2.9 résume l'approche proposée.

TABLE 2.9 – Approche de (Bouchelligua *et al.*, 2010) en synthèse

Modélisation	Modèles de l'approche	- Modèle de concepts - Modèle des tâches - Modèle de workflow - IU Abstraite - IU Concrète - Modèle de contexte
	Conformité au MDA	Oui
	Transformation des modèles	Développement, en Kermeta, des règles de transformation paramétrées dont les paramètres sont des éléments de la plateforme cible, du profil utilisateur et de l'environnement.
Personnalisation	Cible d'adaptation	Adaptation de la présentation du contenu
	Moment d'adaptation	Design time
	Éléments du contexte	≺Utilisateur - Plateforme - Environnement≻
	Modélisation du contexte	Proposition d'un modèle de contexte
	Prise en compte du contexte	La plateforme cible, le profil d'utilisateur et l'environnement représentes les paramètres de la transformation paramétrée, pendant le passage de l'IUA vers l'IUC.
Implémentation	Langage de spécification	XMI
	Outils associés	Kermeta

2.3 Récapitulatif et synthèse

Après avoir présenté quelques approches basées sur les modèles et visant la personnalisation des IHM en fonction du contexte et après avoir détaillé chaque approche à part, nous proposons une synthèse globale permettant de classifier et de comparer les différentes méthodologies traitées.

Cette synthèse se base sur les mêmes critères de départ proposés dans le chapitre 2, §2.1. Dans un premier temps, nous résumons les différents aspects liés à la *modélisation*. Dans un second temps, une comparaison entre ces approches sera établie selon les différents critères liés à la *personnalisation*. Finalement, nous établissons un résumé traitant la partie *Implémentation*.

2.3.1 La modélisation en synthèse

Parmi les méthodes étudiées dans le chapitre 2, §2.2, certaines sont à considérer comme des frameworks (cadres de référence) à suivre lors de la conception d'une approche d'adaptation des IHM basée sur les modèles, comme c'est le cas pour CAMELEON et Dygimes. Bien que le nombre des modèles adoptés par chaque approche et leurs niveaux d'abstraction diffèrent d'une proposition à une autre, nous remarquons que la majorité des approches proposées sont inspirées du framework CAMELEON. Ce cadre de référence leur permet de définir les étapes essentielles pour le développement des IHM sensibles au contexte, à savoir la spécification des concepts et des tâches, la définition de l'interface abstraite, l'interface concrète, et l'interface finale.

Parmi les approches qui suivent CAMELEON on peut citer (Thevenin, 2001; Mori *et al.*, 2004; Limbourg *et al.*, 2005; Sottet *et al.*, 2007) et (Bouchelligua *et al.*, 2010). D'autres approches ont préféré suivre le cadre de référence Dygimes pour définir leurs architectures (ex :(Clerckx *et al.*, 2005)). Nous avons constitué également une troisième classe de propositions, qui ne suivent pas intégralement un framework bien déterminé mais s'en inspirés et ont proposé leurs propres modèles (ex : (Gajos et Weld, 2004; Brossard, 2008; Hachani *et al.*, 2009)).

Dans un objectif de standardisation, certains auteurs on projeté leurs propositions sur une architecture de types MDA (Limbourg *et al.*, 2005; Brossard, 2008; Bouchelligua *et al.*, 2010) tandis que les autres se sont contentés de proposer des approches basées sur les modèles qui ne respectent pas l'architecture de ce paradigme (cf. Figure 2.4). Ce nombre limité d'approche de type MDA est expliqué par la difficulté de découpler les aspects liés à la plateforme depuis les plus hauts niveaux de conception.

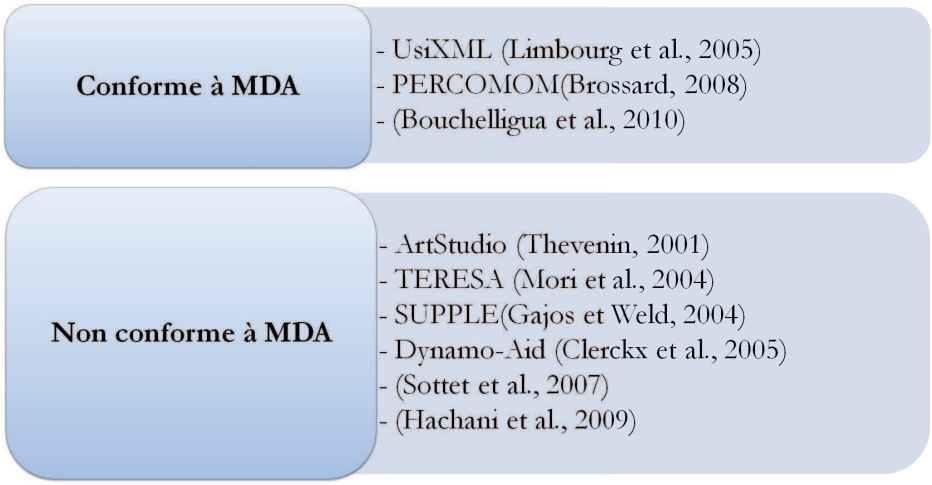


FIGURE 2.4 – Classification des approches selon le conformité à MDA

Le développement des règles de transformation dans une approche basée sur les modèles permet de connaitre son degré de maturité. Dans le panel présenté dans la section précédente, certaines approches n'ont pas implémenté de règles de transformation pour les modèles qu'ils proposent (ex : (Brossard, 2008)). Cependant, dans le cas des approches où les règles de transformations ont été implémentées, la technique de transformation des modèles varie d'une proposition à une autre :

– Utiliser les langages de transformation des modèles tels que ATL (cas de (Sottet *et al.*, 2007) et (Hachani *et al.*, 2009)) et Kermeta (cas de (Bouchelligua *et al.*, 2010)) ;
– Utiliser les techniques de transformation de graphes (cas de (Limbourg *et al.*, 2005)) ;
– Traiter la transformation comme un problème d'optimisation mathématique (cas de (Gajos et Weld, 2004)).

Les autres approches n'ont pas fourni suffisamment de détails sur la manière avec laquelle leurs modèles sont traités et ne proposent pas de dévoiler leurs heuristiques de transformation (cf. Figure 2.5).

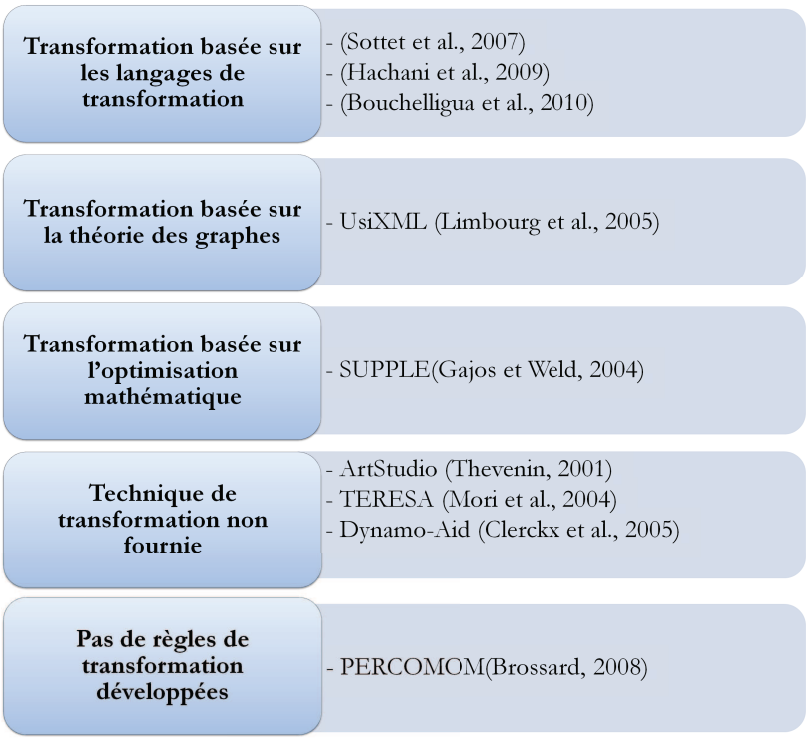


FIGURE 2.5 – Classification des approches selon la transformation des modèles

2.3.2 La personnalisation en synthèse

Comme cité auparavant, la prise en compte du contexte dans la conception des approches basées sur les modèles est une orientation assez récente dans le domaine de la recherche

scientifique. Chaque approche, met en œuvre un ensemble de modèles qui favorisent l'adaptation de l'IHM en fonction du contexte en employant différentes méthodologies.

Ainsi, afin d'implémenter leurs démarches d'adaptation, certains auteurs ont développé leurs propres modèles de contexte à différents niveaux d'abstraction (Thevenin, 2001; Gajos et Weld, 2004; Limbourg *et al.*, 2005; Hachani *et al.*, 2009; Bouchelligua *et al.*, 2010). Selon les besoins et les objectifs de l'approche proposée, la prise en compte du contexte peut s'appuyer sur la totalité des dimensions éléments du contexte ou n'utiliser qu'une seule partie. La Figure 2.6 résume cette classification.

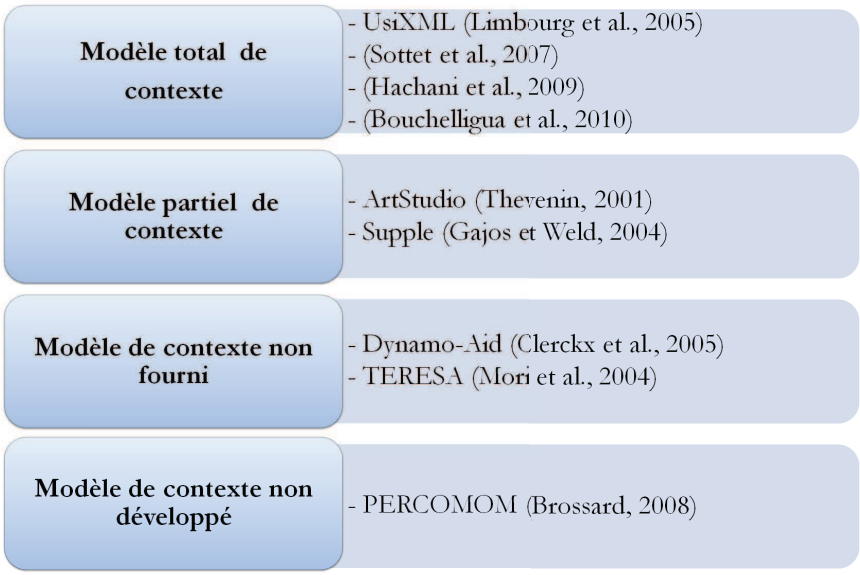


FIGURE 2.6 – Classification des approches selon la modélisation du contexte

Dans les approches basées sur les modèles produisant des IHM sensibles au contexte, certains éléments subissent l'adaptation (ou la personnalisation) tandis que d'autres modèles influencent cette adaptation. D'après l'étude et l'analyse réalisées dans le chapitre 2, §2.2, les auteurs visent majoritairement l'adaptation de la présentation de l'interface et, à notre connaissance, l'unique proposition qui s'intéressait à la personnalisation du contenu était PERCOMOM (Brossard, 2008) (cf. Figure 2.7).

Cependant, la manière d'exploitation des modèles de contexte varie d'une approche à une autre. Par conséquent, certaines propositions se concentrent sur l'annotation du modèle des tâches (ex : (Mori *et al.*, 2004; Clerckx *et al.*, 2005; Brossard, 2008; Hachani *et al.*, 2009)) comme processus central pour l'adaptation, par contre d'autres propositions s'appuient sur la prise en compte des éléments du contexte dans les règles de transformation (ex :(Thevenin, 2001; Gajos et Weld, 2004; Sottet *et al.*, 2007; Bouchelligua *et al.*, 2010)).

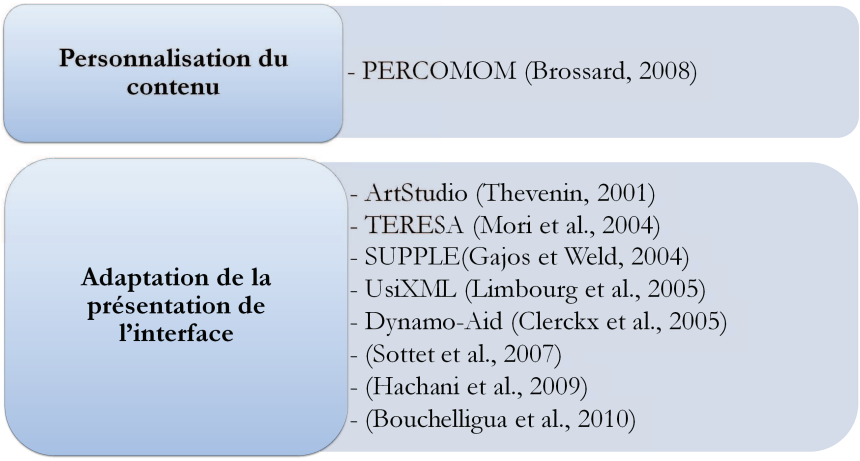


FIGURE 2.7 – Classification des approches selon la cible d'adaptation

2.3.3 L'implémentation en synthèse

D'après l'étude que nous avons faite et comme le montre la Figure 2.8, l'état d'avancement et le degré de maturité varient d'une approche à une autre. Certaines approches sont orientées conception et se contentent seulement de proposer des méthodologies et des outils de modélisation sans avoir recours à l'implémentation de l'approche (ex : (Brossard, 2008)). D'autres approches sont plutôt orientées implémentation en proposant des outils pour spécifier leurs modèles et les mécanismes de transformation entre ces modèles (Sottet *et al.*, 2007; Hachani *et al.*, 2009; Bouchelligua *et al.*, 2010). Elles présentent une palette complète d'outils permettant de prendre en charge les modèles de l'approche de la phase de de modélisation jusqu'à la phase de génération des IHM finales (ex : (Thevenin, 2001; Berti *et al.*, 2004; Gajos et Weld, 2004; Limbourg *et al.*, 2005; Clerckx *et al.*, 2005)).

Afin de mettre en œuvre leurs propositions, les auteurs reposent sur plusieurs notations pour décrire leurs modèles. Certains ont réutilisé des notations existantes avec quelquefois une légère extension, d'autres étaient obligés de proposer de nouveaux formalismes spécifiques à leurs méthodologies.

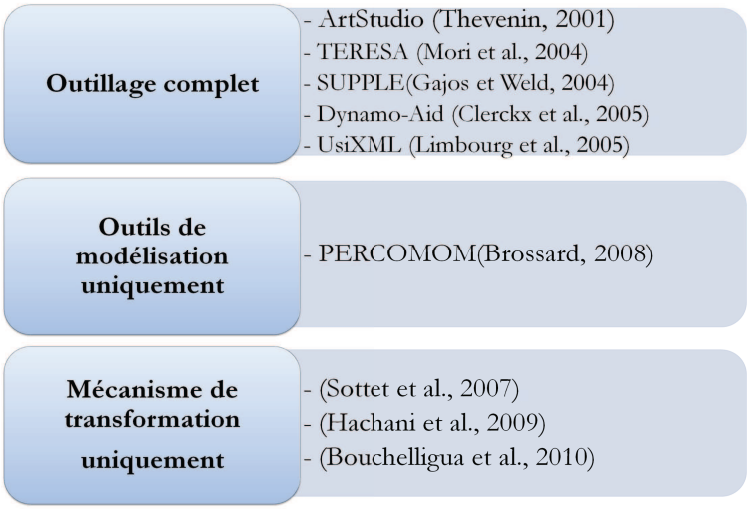


FIGURE 2.8 – Classification des approches selon les phases implémentées

Conclusion

A travers ce chapitre, nous avons présenté un état de l'art sur les approches basées sur les modèles visant la génération des IHM sensibles au contexte. Nous avons commencé par définir l'ensemble de critères à travers lesquels nous avons pu étudier et analyser en détails chaque approche. Cette étude a été suivie d'une synthèse permettant de classifier les propositions étudiées selon différentes perspectives.

Cet état de l'art a révéler l'intérêt et la capacité des approches basées sur les modèles à générer des interfaces personnalisées ou adaptées. Cependant, il a également mis en évidence certaines lacunes de ces approches, notamment dans la prise en compte de la personnalisation du contenu. Généralement, ces approches s'intéressent à l'adaptation des éléments de l'interface (champs, résolution et taille de l'écran, etc.) en se basant sur quelques informations du contexte d'utilisation. Or, pour mettre en œuvre la personnalisation, il est important de considérer non seulement le contenant mais le contenu également.

Dans ce cadre, nous allons essayer, dans le chapitre suivant, de proposer une approche qui s'inscrit dans la lignée d'ingénierie dirigée par les modèles. Notre objectif est d'intégrer la personnalisation du contenu dans la conception des applications interactives en nous basant sur une approche MDA et en tenant compte des informations sur le contexte d'utilisation. Pour atteindre cet objectif, le modèle de contexte et l'ontologie de domaine sont considérés comme éléments centraux de conception et de transformations de modèles.

Deuxième partie

Approche MDA proposée pour l'intégration de la personnalisation du contenu dans la conception et la génération des applications interactives

Chapitre 3

Architecture MDA de l'approche proposée

Introduction

Après la revue de littérature présentée dans le chapitre précédent, nous pouvons affirmer que dans le domaine des IHM, la pertinence des informations fournies et leur adaptation aux préférences des utilisateurs sont des facteurs clés de succès ou de rejet des IHM. De ce fait, la solution est de conquérir les utilisateurs en leur fournissant des systèmes personnalisés et adaptés à leur besoins. Alors que les approches précédemment présentées traitent l'adaptation du contenant, nous cherchons dans ce chapitre à explorer la personnalisation du contenu.

En outre, le Model Driven Architecture (MDA) devient un paradigme important pour le développement des applications. Il peut être appliqué dans plusieurs domaines, étant donné sa capacité à réduire la complexité du processus de développement. Dans cette lignée, nous avons défini une approche de type MDA, permettant de générer des IHM à contenus personnalisés en mettant en œuvre deux méthodes de personnalisation, à savoir le remplissage automatique des formulaires et l'enrichissement des requêtes.

Dans la première section, nous commencerons par décrire notre approche globale. Ensuite, nous détaillerons chaque modèle et la manière avec laquelle il est intégré dans l'approche. Et finalement, nous présenterons les différentes étapes à suivre pour passer d'un niveau MDA à un autre, jusqu'à la génération d'une IHM finale à contenu personnalisé.

3.1 Architecture globale de l'approche

Nous rappelons que notre objectif principal est d'inclure la personnalisation du contenu dès la phase de conception de l'IHM et d'en générer, par la suite l'IHM finale, d'une manière semi-automatique. Afin de répondre à cet objectif et compte tenu de la revue de la littérature présentée dans le chapitre 2, nous avons identifié les exigences suivantes :

a) La mise en œuvre de la personnalisation du contenu.

b) La prise en compte de cette personnalisation dès les premières phases de conception de l'IHM.

c) La génération de l'IHM finale, d'une manière semi-automatique.

En nous basant sur ces exigences, nous avons identifié les différents constituants de notre approche ainsi que ses caractéristiques techniques. En effet, pour pouvoir mettre en œuvre la personnalisation du contenu (besoin (a)), nous avons déterminé, selon les travaux du chapitre 2, la nécessité de prendre en compte l'ensemble des informations et les sources de données sur le *domaine d'application* autours duquel le système a été développé. En plus, il faut considérer également *le contexte* puisque nous cherchons à fournir à l'utilisateur des informations pertinentes et personnalisées. Nous considérons le contexte comme étant le triplet composé de l'utilisateur, la plateforme et l'environnement. Par ailleurs, afin d'assurer la personnalisation du contenu, le domaine ainsi que le contexte doivent être liés étant donné que les informations personnalisées seront livrées selon le contexte. Par

conséquent, nous devons définir *"les relations de dépendance"* qui existent entre le contexte et le domaine d'application. Cette correspondance sera réalisée par le billet du *modèle de mapping*.

Afin de répondre au besoin (b) et pouvoir prendre en compte la personnalisation du contenu dès la phase de conception, nous pouvons avoir recours à un modèle des tâches qui permet de modéliser l'ensemble des interactions. Ce choix fait apparaitre une nouvelle exigence sur la méthode à mettre en place pour inclure la personnalisation du contenu dans le modèle des tâches. À cette fin, il faut d'abord définir les informations du contexte que l'on doit considérer. Et puisque le contenu dépend du contexte, il faut modéliser le domaine ainsi que le contexte et identifier les informations du domaine qui peuvent être influencées par le contexte. Un modèle des tâches est en général composé de l'ensemble des tâches ainsi que les informations du domaine. Dans notre cas, ces informations concernant le domaine d'applications seront spécifiées à travers une ontologie de domaine. Enfin, pour mieux spécifier, dans le modèle des tâches, la manière avec laquelle les informations de contexte doivent être utilisées pour fournir de personnalisation du contenu, il est important de définir, d'une manière abstraite, les *éléments d'interaction* dans lesquels les contenus seront présentés. En effet, la manière avec laquelle le contenu va être présenté peut soutenir la personnalisation de ce contenu.

Enfin, pour répondre à la troisième exigence (besoin (c)), nous décidons d'explorer la piste de l'architecture MDA (OMG, 2003), étant donné sa capacité à générer automatiquement ou semi-automatiquement des IHM à partir d'un ensemble de modèles abstraits. Cette solution est particulièrement intéressante dans le cas de générations des IHM personnalisées pouvant être exécutées sur différents types de plateformes.

En respectant l'ensemble des exigences précédentes, nous proposons une approche de type MDA, permettant la génération semi-automatique des IHM à contenus personnalisés. Cette approche respecte l'architecture de MDA tel qu'elle a été proposée par (OMG, 2003) : le CIM, comportant le modèle des tâches, pour décrire les interactions Homme-Machine dans un haut niveau d'abstraction ; le PIM, pour spécifier l'IHM ainsi que l'ensemble des interactions en utilisant un langage de description d'interface utilisateur ; et le PSM, pour projeter l'interface sur une plateforme spécifique.

Au sein de l'approche proposée, le modèle de contexte, le modèle du domaine et leurs mappings ainsi que le modèle d'interaction, constituent le cœur du processus de personnalisation du contenu. De ce fait, nous les appelons *"les modèles de personnalisation"*. Le CIM est appelé Business Process Model (*BPM*) tandis que le PIM et le PSM sont nommés respectivement Platform Independent Interaction Model (*PIIM*) et Platform Specific Interaction Model (*PSIM*). Dans l'approche proposée, le passage du CIM vers le PIM s'effectue à l'aide d'une transformation de l'ensemble des modèles du niveau CIM. Le passage de PIM vers le PSM est mis en œuvre via l'inclusion ou la modification des caractéristiques spécifiques à la plateforme cible. Par conséquent, nous avons préféré appeler cette étape une "*transition*" au lieu de transformation tandis que le passage du PSM vers le code source est appellé "*génération*" étant donné que nous générons le code de l'IHM finale.

La Figure 3.1 résume l'architecture globale de l'approche proposée. Dans les sections suivantes nous détaillerons les modèles de chaque niveau ainsi que le passage d'un niveau à un autre.

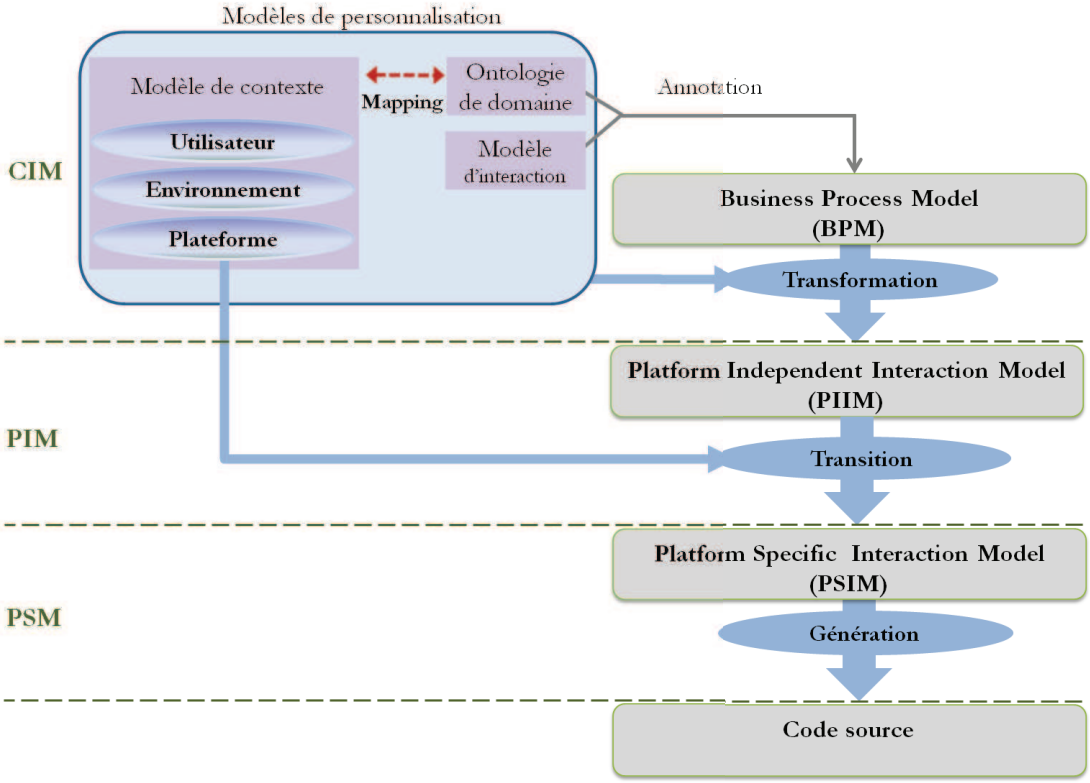


FIGURE 3.1 – Architecture globale de l'approche proposée

3.2 Les modèles de personnalisation

3.2.1 Le modèle de contexte

Tel que mentionné précédemment, notre objectif est de générer des IHM à contenus personnalisés en nous basant sur une approche de type MDA. Afin de permettre une meilleure productivité et de promouvoir la prise en compte du contexte dès les premières phases de conception, il est essentiel d'utiliser un modèle de contexte générique qui peut être utilisé dans la conception de toute interface indépendamment du domaine et de la plateforme d'interaction. Cependant, ce modèle de contexte doit être, au même temps, suffisamment détaillé pour permettre la mise en œuvre de la personnalisation du contenu, et prendre en compte les informations d'un domaine spécifique. La principale difficulté dans la définition

d'un modèle de contexte est de trouver un compromis entre la *généralité* et la *spécificité* du modèle de sorte que le modèle proposé soit, d'une part, réutilisable dans la conception de plusieurs interfaces utilisateur, d'autre part, adapté à différents domaines d'application.

Notre première idée était de réutiliser un modèle de contexte existant. Pour cela, nous avons effectué une revue de la littérature de 18 propositions de modélisation de contexte. Ensuite, nous les avons classées en trois groupes, compte tenu des dimensions du contexte telles qu'elles ont été définies par (Calvary *et al.*, 2003), à savoir l'utilisateur, la plateforme et l'environnement.

Le premier groupe se réfère aux propositions qui ne tiennent compte que d'une seule dimension de contexte. Ce sont les modèles contextuels qui mettent l'accent sur l'*utilisateur*, son profil, ses intérêts, etc (UMO, 2003; Rousseau *et al.*, 2004; Kostadinov, 2008). Il existent également ceux qui détaillent les *platesformes* (FIPA, 2001) et leurs relations avec l'environnement (W3C, 2009a), ou bien qui mettent l'accent sur l'*environnement*, compte tenu des informations sur la localisation de l'utilisateur (Becker et Dürr, 2005) et le temps (Hobbs et Pan, 2006).

Le second groupe retient les propositions qui considèrent la totalité des dimensions mais qui sont dédiées à un domaine spécifique ou une technologie particulière. Ces modèles doivent être détaillés pour répondre aux besoins spécifiques d'un domaine bien déterminé. Dans ce groupe, nous pouvons citer les modèles proposés dans les domaines de l'informatique ubiquitaire (Chen *et al.*, 2004; Lin *et al.*, 2005), des maisons intelligentes (Kim et Choi, 2006), des applications mobiles (Schmidt *et al.*, 1999; Korpipaa *et al.*, 2003; Weibenberg *et al.*, 2004) et des applications de commerce électronique (Taconet et Aoul, 2008).

Enfin, le troisième groupe, s'appuie sur les modèles qui sont indépendants du domaine, et qui contiennent les trois dimensions de contexte (l'utilisateur, l'environnement et la plateforme). Cependant, ces modèles ne sont pas bien détaillés. Par exemple, (Preuveneers *et al.*, 2004) incluent la définition d'un profil de l'utilisateur dans la dimension utilisateur, mais ne définissent pas les éléments à considérer dans ce profil. (Wang *et al.*, 2004) et (Arabshian et Schulzrinne, 2006) ont proposé des ontologies de haut niveau d'abstraction, qui devraient être associées à un domaine d'intérêt spécifique. Enfin, dans le cadre du projet UsiXML (Limbourg *et al.*, 2005; UsiXML, 2007), les auteurs explorent la dimension plateforme plus que les autres dimensions, étant donné qu'ils visent principalement, l'adaptation des IHM. Leur modèle de l'environnement prend en compte seulement trois aspects (la lumière, le bruit et le stress). Le premier modèle de l'utilisateur proposé dans le cadre de UsiXML est limité à quelques attributs qui décrivent l'expérience de l'utilisateur avec la plateforme. Il a été récemment modifié par (Tesoriero et Vanderdonckt, 2010), pour tenir compte des éléments qui influencent une interface utilisateur. Cela se fait selon deux niveaux d'abstraction : un niveau des éléments où le concepteur définit les caractéristiques des utilisateurs qui sont pertinents pour le domaine d'application ; un niveau profil qui décrit ces caractéristiques en fonction des situations rencontrées lors de l'exécution. Bien que cette méta-modélisation offre une flexibilité dans la définition de la dimension utilisateur, elle nécessite encore un effort supplémentaire pour définir toutes les

caractéristiques pour chaque nouveau domaine d'application.

Toutefois, tel que présenté précédemment, nous pouvons remarquer que les modèles de contexte proposés sont soient très génériques d'une manière à rendre difficile la personnalisation du contenu, soit très spécifiques pour un domaine particulier ou pour une unique dimension de contexte. Ceci rend difficile la réutilisation de ces modèles pour différentes applications.

Par conséquent, nous avons jugé meilleur de définir notre propre modèle de contexte, en tenant compte des propositions précédemment présentées. Pour élaborer cette modélisation, nous avons considéré trois étapes nécessaires :

1. Analyser tous les concepts et les propriétés des 18 modèles de contexte étudiés.
2. Classer ces concepts et propriétés dans des catégories en fonction de leurs sens sémantiques.
3. Organiser ces catégories autour des trois dimensions principales du contexte : l'utilisateur, la plateforme et l'environnement.

Étant donné que les modèles que nous utilisons pour élaborer notre proposition sont définis en anglais, les concepts et les attributs de notre modèle de contexte seront exprimés en anglais.

Les sous-sections qui suivent présentent les modèles d'utilisateur, de la plateforme, et de l'environnement.

3.2.1.1 Le profil de l'utilisateur

En analysant les concepts de la littérature étudiée, nous organisons le profil de l'utilisateur en cinq grandes catégories qui décrivent l'utilisateur lors de son interaction avec la plateforme (cf. Figure 3.2) : *demographic information*, *contact information*, *user preferences*, *user state* et *user abilities & proficiencies*.

Dans la littérature, certains auteurs (Kostadinov, 2008; UMO, 2003) distinguent entre *Contact information* et *User demographic data* et d'autres (Rousseau *et al.*, 2004; Lin *et al.*, 2005; Preuveneers *et al.*, 2004) les composent. Par souci de clarté, et en suivant la première lignée, nous avons défini les classes suivantes :

1. La classe *Contact information* contient les données personnelles de l'utilisateur. Ces données peuvent changer au cours du temps ;
2. La classe *Demographic information* contient les données de base de l'utilisateur qui ne changent pas généralement ;
3. La classe *Preference* représente les préférences de l'utilisateur ainsi que ses intérêts. Certaines approches utilisent uniquement le terme *preference* (Kostadinov, 2008; Preuveneers *et al.*, 2004; UMO, 2003), tandis que d'autres (Rousseau *et al.*, 2004) utilisent le terme *interest*. Bien qu'il y ait la possibilité d'utiliser des types complexes pour exprimer ces préférences tels que ceux proposés par (Kostadinov, 2008), nous

avons choisi les préférences unitaires et simples, composées d'un seul attribut avec un type *booléen* afin d'indiquer si l'utilisateur préfère ou pas un élément spécifique ;

4. La classe *User State* présente l'état de l'utilisateur lors de l'interaction avec l'IHM. D'après la littérature, cet état peut être émotionnel, physiologique (Schmidt *et al.*, 1999; UMO, 2003) ou peut être une activité pratiquée par l'utilisateur (Kim et Choi, 2006; Korpipaa *et al.*, 2003). Nous n'avons pas inclus l'état émotionnel dans notre modèle, car il est rarement utilisé pour décrire l'utilisateur ;
5. Et enfin, la classe *Ability & Proficiency* permet de décrire l'ensemble de connaissances de l'utilisateur, de ses compétences et de ses capacités. Cette classe est une adaptation de celle proposée dans (UMO, 2003).

Le Tableau 3.1 représente l'ensemble des concepts identifiés dans les 18 propositions étudiées, répartis sur les cinq classes décrivant le profil de l'utilisateur.

TABLE 3.1: Les concepts identifiés caractérisant le profil utilisateur

Classe	**Concepts**	**Référence**
Contact information	Family name, address, e-mail, phone / fax number	(Kostadinov, 2008)
	Name	(Kostadinov, 2008), (Weibenberg *et al.*, 2004)
	Contact Information/detail (city, country, email, family name, fax/ phone number, full name, postal code, street)	(UMO, 2003), (Rousseau *et al.*, 2004)
Demographic information	Date of birth, occupation, children, revenue, marital status	(Kostadinov, 2008)
	Demographics (age, age group, birthday, birthplace, salary, employment, family status, first language, gender, wealth)	(UMO, 2003)
	Gender	(Lin *et al.*, 2005), (Kostadinov, 2008)
	Affiliation	(Rousseau *et al.*, 2004)
Preferences	Preference (interface preference, privacy preference)	(UMO, 2003)
	Movie preference, music preference, news preference	(Kim et Choi, 2006)
	Preference profile	(Preuveneers *et al.*, 2004)
	Simple preference, complex preference	(Weibenberg *et al.*, 2004), (Kostadinov, 2008)
	Interest	(UMO, 2003), (Rousseau *et al.*, 2004)
	Interests (Olympic, shopping, sightseeing, entertainment)	(Weibenberg *et al.*, 2004)

Ability & Proficiency	First Language, Second Language, Knowledge, Computer skills, Reading skills, writing skills, typing skills	(UMO, 2003)
	Career	(Lin *et al.*, 2005)
	Language read – language spoken – language written	Weibenberg *et al.* (2004)
	Competency (Skills, knowledge), Qualifications	(Rousseau *et al.*, 2004)
	Habits	(Schmidt *et al.*, 1999)
	Author, developer, learner, reader, teacher, user, ...	(UMO, 2003)
	Profession	(Kostadinov, 2008)
	Abilities, disabilities	(Rousseau *et al.*, 2004)
	Ability and Proficiency (ability to talk, to drive, to hear, to see . . .)	(UMO, 2003)
User State	Physiological State (blood pressure, injury, respiration, temperature , ...)	(UMO, 2003)
	Motion (lying, going up stairs, sitting, standing, walking)	
	Emotional state (anger, anxiety, disgust, happiness, sadness)	
	Mental State (depression – irritation – nervousness – psychopathy – trauma)	
	Emotional state - biophysiological conditions	(Schmidt *et al.*, 1999)
	Activity (Sleeping, Watching TV, Cleaning, Getting Up)	(Kim et Choi, 2006), (Preuveneers *et al.*, 2004), (Weibenberg *et al.*, 2004), (Rousseau *et al.*, 2004), (Korpipaa *et al.*, 2003), (Wang *et al.*, 2004)
	Mood	(Preuveneers *et al.*, 2004)

D'après le tableau précédent, nous pouvons constater que certaines classes du profil utilisateur peuvent être communes à tous les domaines (*Contact information* et *Demographic information*) tandis que d'autres dépendent directement du domaine d'application et ne peuvent pas être génériques (*Preferences*, *Ability & Proficiency* et *User State*). De ce fait, les concepts relevés ne peuvent pas être exhaustifs pour couvrir la totalité des domaines et nous devrons donc généraliser la dimension utilisateur dans un méta-niveau d'abstraction qui pourrait être instancié et prêt à l'utilisation dans la modélisation des tâches.

En tenant compte de cet aspect, les classes *Preference,User State* et *Ability & Proficiency* seront présentées sous la forme de métamodèles qui doivent être instanciés pour chaque

domaine avant de commencer la conception des IHM. La Figure 3.2 montre notre modèle de profil utilisateur proposé. Il contient deux parties : une partie *instanciable* (M1) qui dépend du domaine d'application et une partie *statique* (M2) applicable à tout domaine.

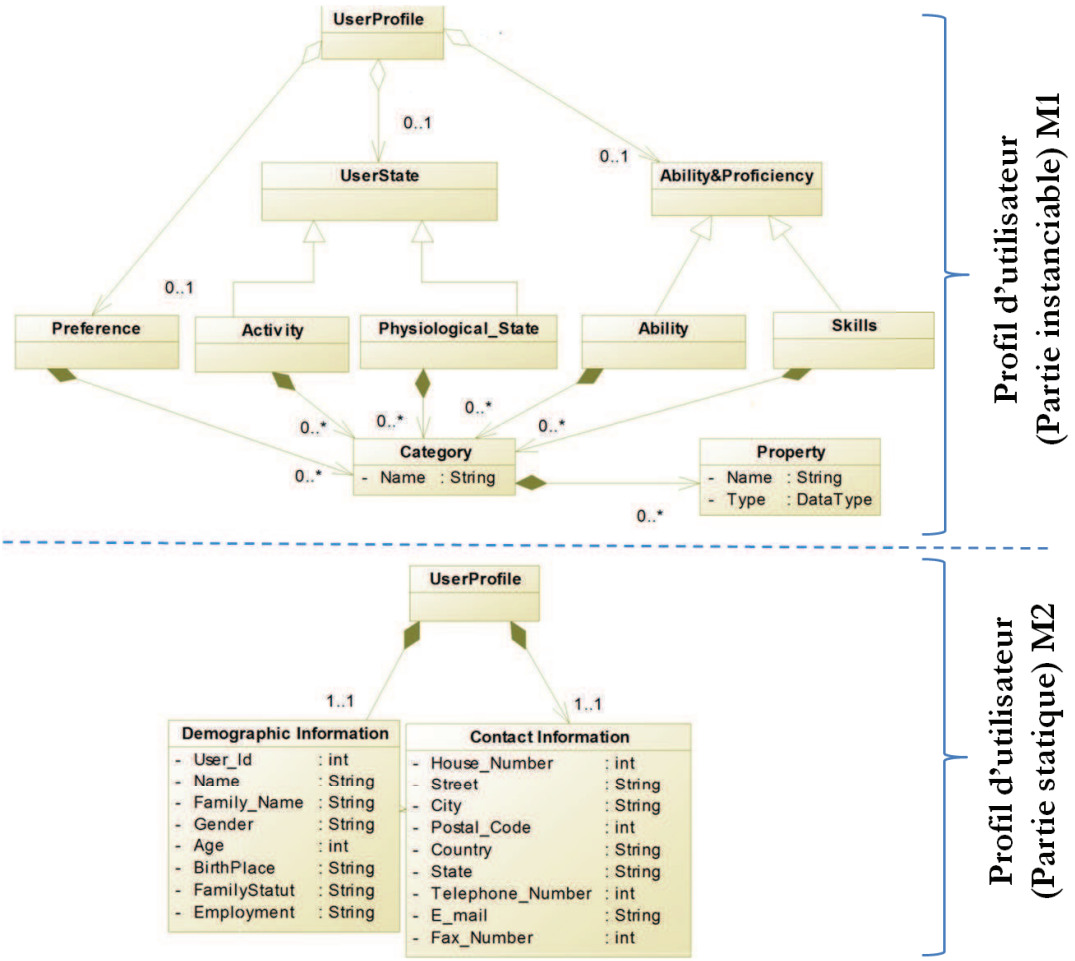


FIGURE 3.2 – Le profil utilisateur proposé

Afin d'obtenir le modèle utilisateur final, nous devons d'abord instancier la partie *instanciable*, selon le domaine d'application. La Figure 3.3, introduit un exemple d'instanciation du modèle M1 permettant de définir un modèle de profil utilisateur spécifique au domaine du transport en commun.

Ensuite, nous devons associer la partie obtenue avec la partie statique. Le modèle final obtenu permet de décrire le contexte pour un domaine spécifique (dans le cas de cet exemple, le domaine de transport). Dans la Figure 3.4, nous présentons une partie de ce modèle. Par souci de clarté, nous avons simplifié la notation précédente. En effet, les

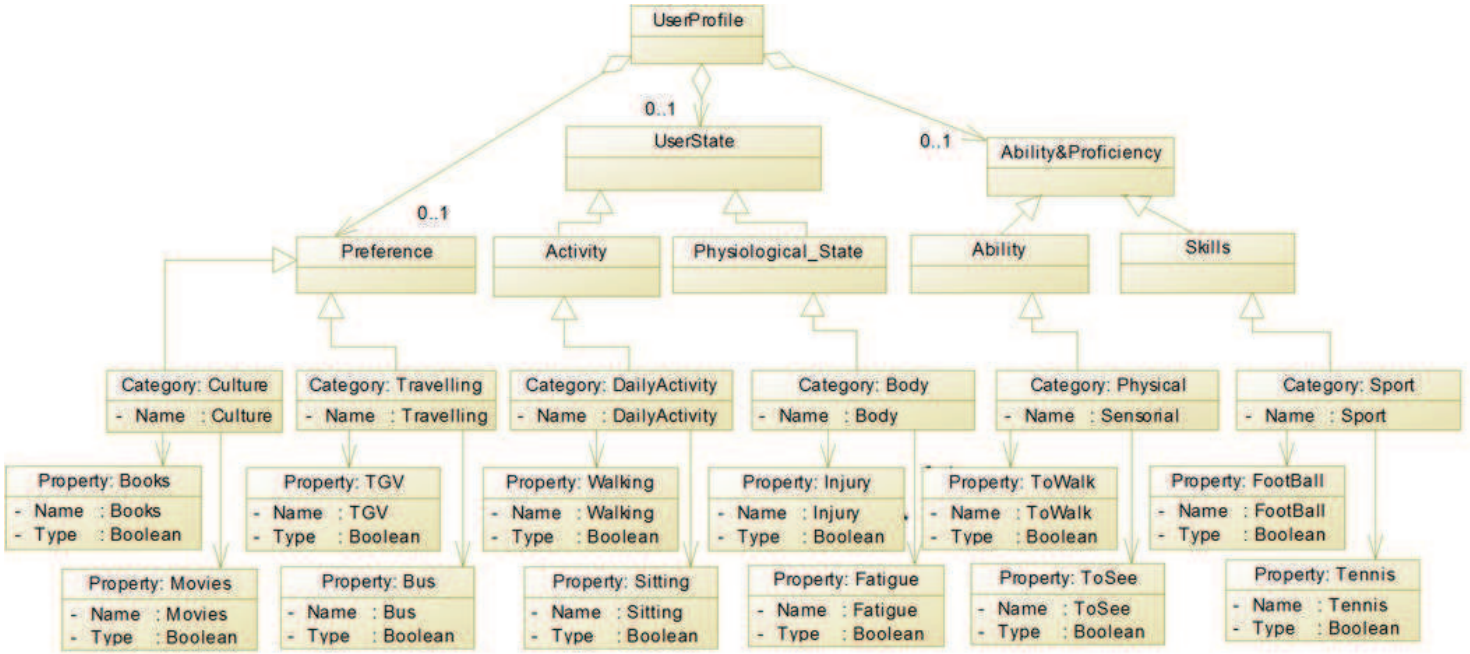


FIGURE 3.3 – Exemple d'instanciation du profil utilisateur

catégories sont modélisées par une classe et les propriétés de celles ci sont représentées comme des attributs.

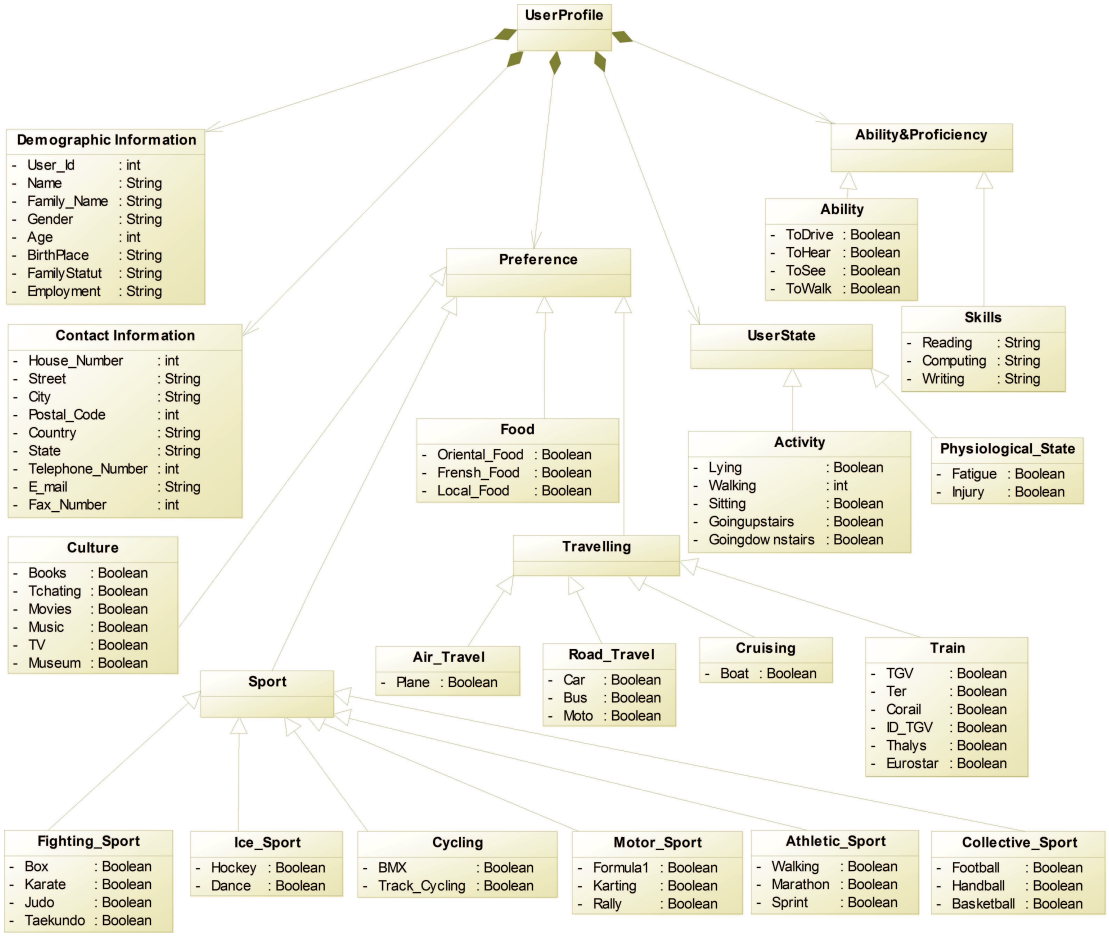


FIGURE 3.4 – Un profil utilisateur spécifique au domaine des transport en commun

3.2.1.2 La plateforme

Cette dimension est nécessaire car elle décrit la plateforme que l'utilisateur utilise pour interagir avec l'IHM. D'après (FIPA, 2001; Preuveneers *et al.*, 2004; UsiXML, 2007; Kostadinov, 2008; W3C, 2009a), la classification usuelle pour décrire une plateforme est de différencier entre la partie matérielle (*hardware*) et la partie logicielle (*software*), comme présenté dans les propositions étudiées dans le Tableau 3.2.

TABLE 3.2 – Les concepts identifiés caractérisant la plateforme

Catégorie	**Concepts**	**Référence**
Software	Operating system (Win Mobil, Symbian, Android)	(Taconet et Aoul, 2008)
	API, RuntimeEnvironment	(W3C, 2009a)
	OS (name-vendor-version)	(FIPA, 2001), (Preuveneers *et al.*, 2004), (UsiXML, 2007)
	Software (name, edition, version), virtual machine, middleware, rendering engine, operating system	(Preuveneers *et al.*, 2004)
Hardware	Memory, CPU	(Taconet et Aoul, 2008), (FIPA, 2001), (Preuveneers *et al.*, 2004), (W3C, 2009a)
	Connection	(Taconet et Aoul, 2008), (FIPA, 2001)
	Display (resolution)	(Taconet et Aoul, 2008)
	Keyboard type (Numeric, qwerty , Touch screen)	
	Network Interface (3G, WIFI, Bluetooth)	
	UI-screen (width-height-unit-resolution-color)	(FIPA, 2001)
	Connection (information-QOS information)	
	Network	(Lin *et al.*, 2005), (Preuveneers *et al.*, 2004), (Wang *et al.*, 2004)
	Resource (Power – memory – CPU – storage – network)	(Preuveneers *et al.*, 2004)
	File format	(Kostadinov, 2008)
	NetworkEntity (NetworkMode – NetworkSupport – NetworkTechnology)	(W3C, 2009a)
	Screen Width – Screen Hight – Screen Size Char - Max Screen Char - Is image capable - Pointing device – Has Touch Screen - Storage Capacity	(UsiXML, 2007)
	Surrounding resources for computation	(Schmidt *et al.*, 1999)

En nous basant sur l'ensemble des travaux présentés dans le Tableau 3.2, nous avons défini notre modèle de la plateforme (cf. Figure 3.5) qui se compose principalement des

deux parties matérielle (*hardware*) et logicielle (*software*). La partie matérielle décrit tous les aspects physiques de la plateforme et elle est composée de quatre sous-parties telles qu'elles sont définies par (Taconet et Aoul, 2008), (FIPA, 2001) et (Preuveneers *et al.*, 2004) :

- *Memory* : pour spécifier la taille de la RAM (Random Acess Memory) de la plate-forme.
- *CPU* : pour représenter le processeur embarqué dans la plateforme ainsi que sa vitesse. Cette information peut être utile dans le but de savoir si la plateforme cible peut exécuter ou non l'interface utilisateur.
- *Network* : pour fournir des informations générales sur les caractéristiques du réseau installé sur la plateforme. Nous pouvons exploiter cette information pour déterminer si la plateforme a la capacité d'être mobile ou non (dans le cas d'une connexion Wi-Fi, par exemple).
- *User interface* : pour indiquer la diemnsion (hauteur et largeur) de l'interface utilisateur ainsi que sa résolution d'image. Ce type de fonctionnalité est important à prendre en compte étant donné qu'elle agit directement sur l'adaptation de la taille des éléments de l'interface.

La partie logicielle de la plateforme se compose de quatre sous-parties telles qu'elles sont définies par (Preuveneers *et al.*, 2004) :

- *Virtual machine* : pour décrire l'ensemble des environnements d'exécution que contient la plateforme. Nous pouvons exploiter cette information afin d'avoir un code portable (par exemple, Java), où il est important de savoir si la plateforme contient la machine virtuelle adaptée pour l'exécuter (par exemple, Java VM).
- *Application system* : pour spécifier l'ensemble des applications installées sur la plateforme.
- *Operating System (OS)* : pour introduire le système d'exploitation avec lequel la plateforme fonctionne. Cette information est essentielle afin de vérifier la compatibilité avec l'application puisque certaines bibliothèques du système d'exploitation pourraient être nécessaires pour l'exécution de certains programmes.
- *Rendering engine* : pour décrire le moteur du rendu qui peut interpréter un code source pour générer des interfaces finales appropriées. Pour le modèle que nous proposons, de telles informations font partie des plus importantes, étant donné que nous travaillons dans le cadre de l'IDM, où la plateforme cible doit interpréter le code source généré automatiquement.

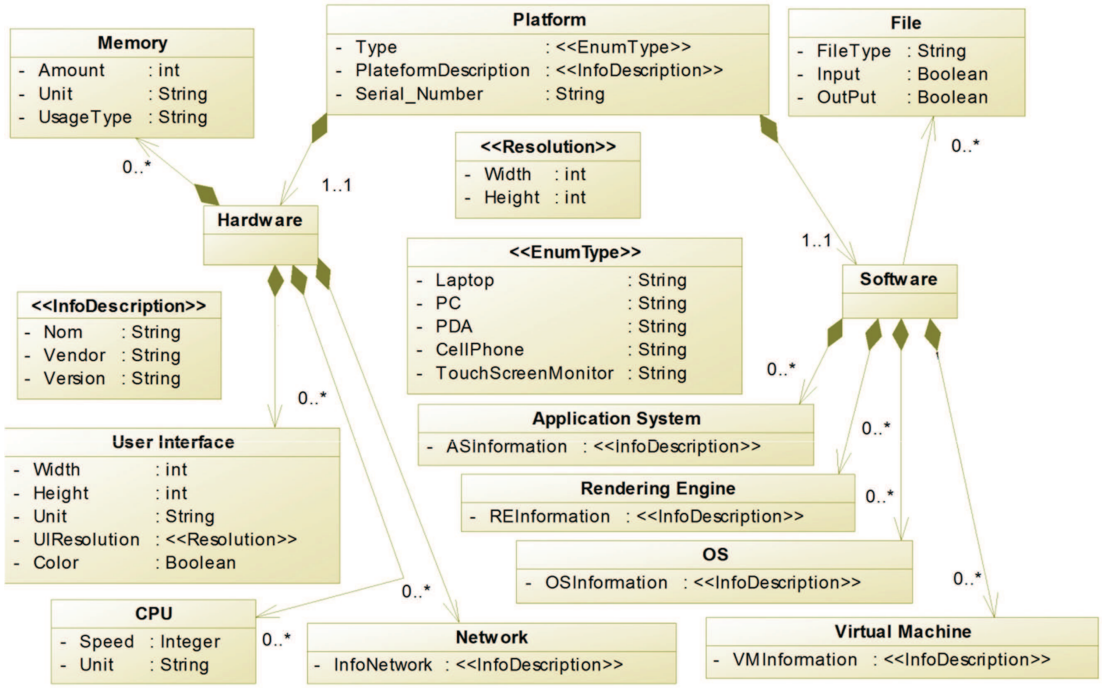


FIGURE 3.5 – Modèle de plateforme proposé

3.2.1.3 L'environnement

Cette dimension du contexte décrit toutes les informations sur l'environnement dans lequel l'interaction entre l'utilisateur et la plateforme se fait. La plupart des informations liées à l'environnement sont dynamiques et peuvent avoir un impact sur le contenu présenté à l'utilisateur. L'état de l'art traitant cette dimension (Korpipaa *et al.*, 2003; Preuveneers *et al.*, 2004; Arabshian et Schulzrinne, 2006; Kostadinov, 2008) détermine plusieurs concepts qui permettent de caractériser l'environnement. Le Tableau 3.3 résume l'ensemble des concepts identifiés pour décrire la plateforme.

TABLE 3.3 – Les concepts identifiés caractérisant l'environnement

Catégorie	**Concepts**	**Référence**
Location	Location (IndoorSpace – OutdoorSpace)	(Wang *et al.*, 2004)
	Geometric (GPS), Symbolic	(Becker et Dürr, 2005)
	Building {Indoors, Outdoors}, GPS Location	(Korpipaa *et al.*, 2003)
	Country, City, zip code, longitude, latitude, coordinates	(Arabshian et Schulzrinne, 2006)
	Geographical place (street, city, province, country)	(Kim et Choi, 2006)
	Geocordinates, UTMCoordinates, WGS84Coordinates, Geographical Coordinate Reference System	(W3C, 2009a)
	Relative, absolute	(Preuveneers *et al.*, 2004)
	Location (absolute position, relative position, co-location)	(Schmidt *et al.*, 1999)
Time	Time	(Arabshian et Schulzrinne, 2006), (Korpipaa *et al.*, 2003)
	TemporalEntity, Interval, Instant, DurationDescription, TemporalUnit	(Hobbs et Pan, 2006)
Environmental Condition	Sound : Intensity {Silent, Moderate, Loud}	(Korpipaa *et al.*, 2003)
	Light : Intensity {Dark, Normal, Bright}	
	Light : Type {Artificial, Natural}	
	Light : Source Frequency {50Hz, 60Hz, Not Available}	
	Temperature {Cold, Normal, Hot}	
	Humidity {Dry, Normal, Humid}	
	Sound : Type {Car, Elevator, Rock Music, Classical Music, Tap Water, Speech, Other Sound}	
	Temperature Value	(Taconet et Aoul, 2008), (Kim et Choi, 2006), (Preuveneers *et al.*, 2004)
	Lighting	(Lin *et al.*, 2005), (Preuveneers *et al.*, 2004)
	Noise	
	IsNoisy – IsStressing – LightingLevel	(UsiXML, 2007)
	Humidity – Pressure - Environmental condition	(Preuveneers *et al.*, 2004)
	Physical conditions (noise, light, pressure, temperature, acceleration)	(Schmidt *et al.*, 1999)

Sur la base de ces informations, nous avons proposé le modèle de l'environnement suivant (cf. Figure 3.6). L'analyse de la littérature nous a permis d'identifier deux classes principales :

- La première, nommée *location*, se réfère à l'endroit où se trouve l'utilisateur au moment de l'interaction. Cet endroit peut être décrit d'une manière déterministe grâce à l'utilisation des données géométriques (telles que les coordonnées GPS, ville, rue, etc) ou par le biais par rapport données symboliques vers un autre emplacement géométrique (*geometric*) (en face, à côté de, ...) tel que proposé dans (Becker et Dürr, 2005). (Li *et al.*, 2007), affirment que parmi les défis auxquels font face les concepteurs des applications sensibles au contexte, on trouve le changement de l'environnement d'une manière dynamique.
- La seconde considère le temps (*time*), qui décrit le moment de l'interaction avec la plateforme (Arabshian et Schulzrinne, 2006; Korpipaa *et al.*, 2003; Hobbs et Pan, 2006). Par analogie avec la localisation, l'heure pourrait être décrite d'une manière exacte (*exact time*) (année, mois, jour, etc) ou (*symbolic time*) d'une manière symbolique (par exemple l'été, les vacances scolaires, etc).

Mis à part le lieu et l'heure de l'interaction, certains auteurs (Korpipaa *et al.*, 2003; Lin *et al.*, 2005; Preuveneers *et al.*, 2004; Kim et Choi, 2006; Schmidt *et al.*, 1999) incluent des informations supplémentaires pour décrire l'environnement (comme la météo, bruits, etc). Cette information supplémentaire liée à l'environnement a été intégrée dans notre modèle sous forme d'une classe nommée *Environmental Condition*.

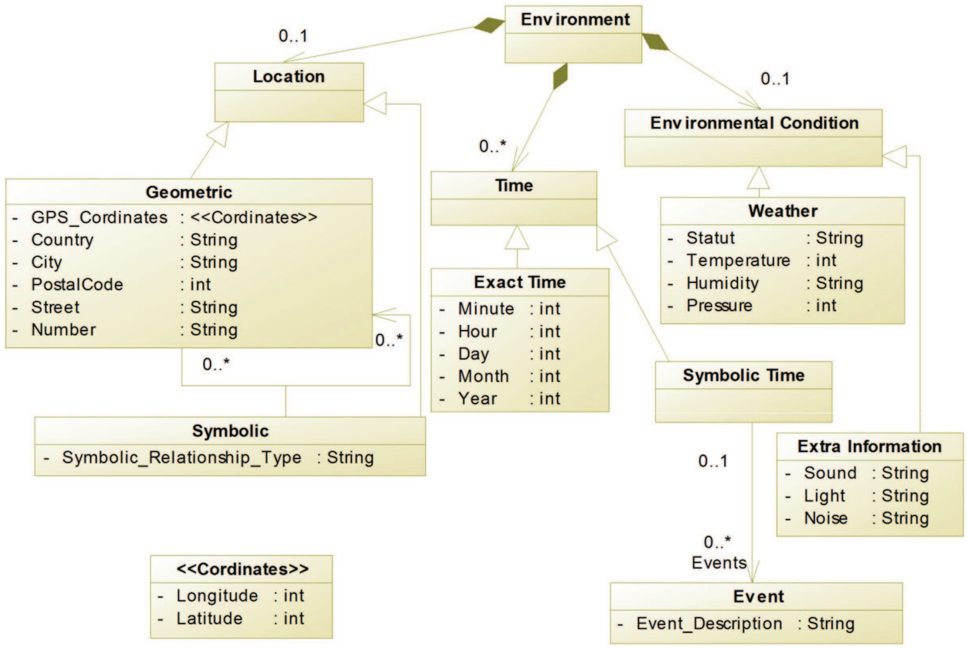


FIGURE 3.6 – Modèle de l'environnement proposé

3.2.2 Le modèle de mapping

Une fois le modèle de contexte défini, il est important de définir de quelle façon il sera utilisé durant la phase de conception des IHM, pour mettre en œuvre la personnalisation du contenu. Nous pensons que le domaine d'application est intimement lié au contexte. Par exemple, une interface utilisateur nécessitant un ensemble d'informations sur l'utilisateur peut avoir recours au modèle de contexte utilisé par le système (à partir du profil de l'utilisateur inclus dans le modèle de contexte).

La question à laquelle il faut répondre est de savoir comment on peut lier le modèle de contexte à un domaine d'application spécifique. Nous rappelons que le modèle du domaine représente le vocabulaire utilisé pour définir l'ensemble des entrées/sorties à travers l'interface utilisateur. Comme nous l'avons décrit précédemment dans le chapitre 1, le modèle de domaine peut être défini à travers un diagramme de classes, un modèle entité-association, ou à travers une ontologie de domaine. Dans le cadre de cette thèse, nous avons choisi de définir le vocabulaire de l'application à travers l'utilisation des ontologies du domaine en raison de leurs capacités à caractériser la sémantique du domaine.

Étant donné que le domaine est décrit à travers une ontologie, nous nous sommes penchés à chercher un moyen pour identifier une relation entre les éléments de l'ontologie et ceux du modèle de contexte. Pour remédier à ce problème et à partir d'une analyse de ces éléments, nous avons établi un ensemble de correspondances entre les deux modèles, que nous avons incarnées dans un modèle, appelé le *modèle de mapping*. Le métamodèle définissant ce modèle de mapping est décrit dans la Figure 3.7. Nous avons défini trois types de mappings :

- **Mapping direct** : lorsqu'un concept de l'ontologie représente exactement la même information présente dans le modèle de contexte, sachant que parfois il peut être exprimé avec un nom différent. Cela signifie que l'élément du contexte et l'élément de l'ontologie de domaine ont la même signification sémantique. Cette correspondance est faite pour n'importe quel attribut dans le modèle de contexte qui n'est pas de type *booléen* (par exemple, l'âge, le nom de l'utilisateur, etc) ;
- **Mapping indicatif** : lorsque les informations du contexte indiquent la présence ou l'absence de certaines informations. Ce type de mapping est similaire au mapping direct, sauf que dans ce cas, l'attribut du contexte doit avoir le type *booléen* ;
- **Mapping indirect** : lorsque certains éléments du modèle de contexte pourraient avoir une influence indirecte sur un élément de domaine. Pour définir un mapping indirect, le concepteur doit vérifier s'il y a des informations dans le modèle de domaine qui pourrait changer en fonction des données modélisées dans le contexte.

Nous notons que les mappings s'appliquent entre les concepts ou les attributs de l'ontologie de domaine d'une part, et entre les attributs du modèle de contexte d'autre part, et qu'il peut y avoir des éléments (concepts ou attributs) qui ne sont dotés d'aucun mapping. Nous signalons également la possibilité de réutilisation des mappings, étant donné qu'une ontologie de domaine est généralement utilisée pour plusieurs applications dans le même domaine. Des exemples de mappings seront présentés dans le chapitre 5.

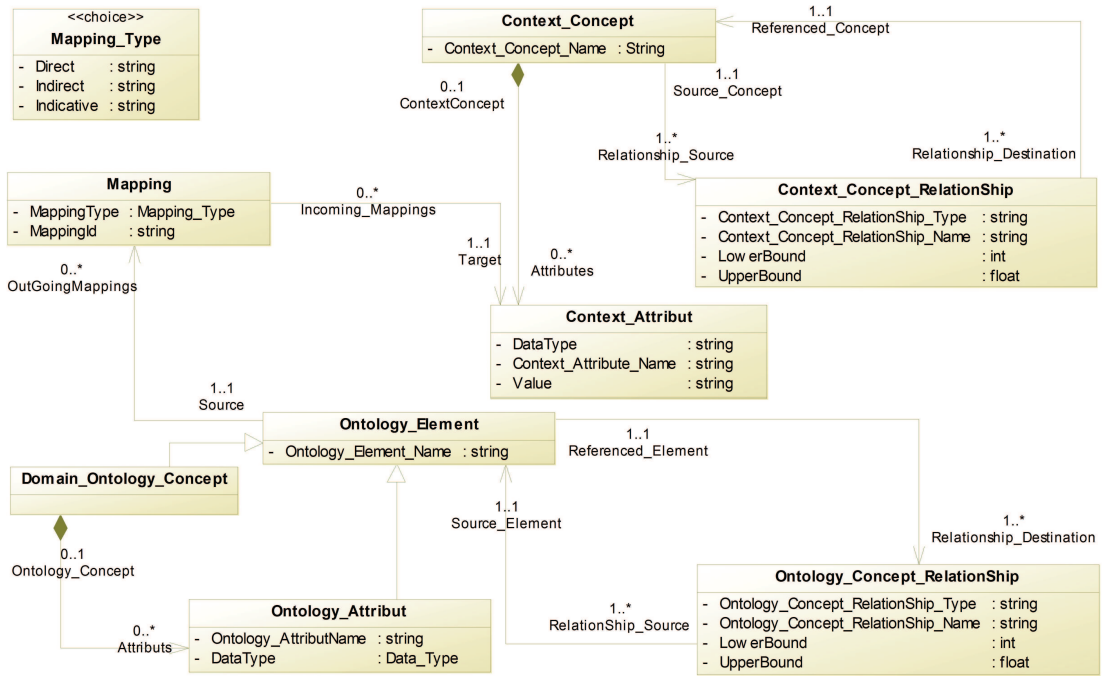


FIGURE 3.7 – Métamodèle de mapping entre le contexte et l'ontologie de domaine

Dans ce qui suit, nous nous référons un élément de l'ontologie (concept ou attribut) par "o" et un attribut du modèle de contexte par "c". Afin de créer un modèle de mapping le concepteur pourrait suivre la logique suivante :

. Si "o" et "c" ont le même sens sémantique et le type de "c" est différent de *Booléen* alors créer un mapping direct entre "o" et "c".

. Si "o" et "c" ont le même sens sémantique et le type de "c" est *Booléen* alors créer un mapping indicatif entre "o" et "c".

. Si la valeur de "o" peut changer en fonction de la valeur de "c" alors créer un mapping indirect entre "o" et "c".

3.2.3 Le modèle d'interaction

Comme indiqué dans le chapitre 3, §3.1, étant donné que nous nous intéressons à la personnalisation du contenu d'une manière semi-automatique, il est important de prendre en compte la manière avec laquelle l'information sera présentée et par conséquent considérer les éléments d'interaction lors de la conception des IHM. En effet, la manière avec la quelle l'information va être présentée pourrait prendre de nombreuses formes et elle est totalement dépendante du type de l'élément d'interaction.

Pour cette raison, nous avons proposé le modèle d'interaction (cf. Figure 3.8), qui représente une abstraction des éléments d'interaction pouvant être utilisés lors de la conception des IHM d'une manière indépendante de toute plateforme d'interaction. Ce modèle a été définit en se basant sur la proposition de (Brossard *et al.*, 2007) à la quelle nous avons apporté quelques extensions.

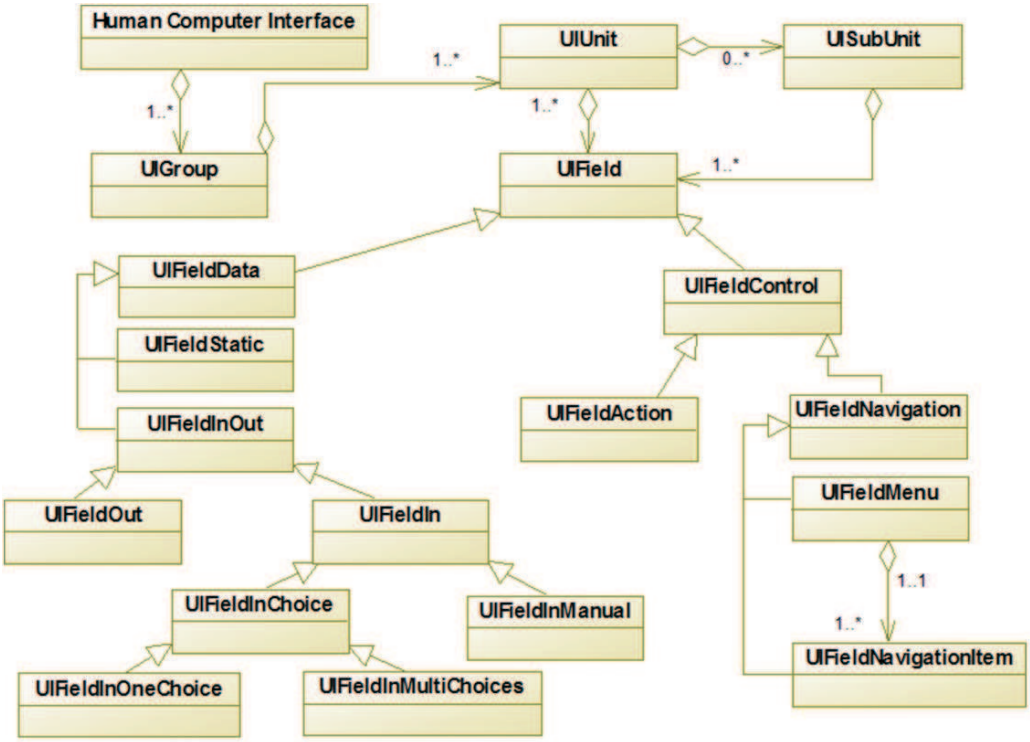


FIGURE 3.8 – Modèle d'interaction

Ces éléments d'interaction sont groupés sous le type *UIGroup* qui représente un groupement logique des éléments d'interaction. Ces groupes contiennent des unités d'interaction, nommées *UIUnit*. De manière similaire, les UIUnits sont constitués des UIElements qui sont des éléments d'interaction élémentaires comme les éléments de navigation, les éléments d'Entrée/Sortie, etc. Ces éléments d'interaction sont préfixés par le terme *UIField*. Un UIField est une abstraction d'un élément de l'IHM permettant à l'utilisateur d'interagir avec celle ci. Tous ces éléments sont supposés être réutilisables par plusieurs applications à fin de permettre la création des applications à travers leur assemblage. Le Tableau 3.4 présente la définition des éléments du modèle d'interaction.

TABLE 3.4 – Les éléments du modèle d'interaction

Élément d'interaction	**Spécification**
UIGroup	Représente l'espace de travail qui permet de grouper logiquement des UIUnits et des UISubUnits.
UIUnit	Permet de regrouper des éléments d'interaction en groupes logiques du point de vue des interactions
UIFieldInManual	Représente un élément d'interaction permettant à l'utilisateur d'effectuer une entrée manuellement.
UIFieldStatic	Représente un élément d'interaction capable d'afficher des informations statiques ne pouvant pas être changées.
UIFieldOneChoice	Définit un élément d'interaction permettant à l'utilisateur de sélectionner ou pas une alternative unique.
UIFieldMultiChoices	II s'agit d'un élément d'interaction permettant à l'utilisateur de sélectionner une alternative parmi plusieurs.
UIFieldOut	Constitue un élément d'interaction permettant au système d'effectuer une sortie d'informations.
UIFieldAction	Est un élément d'interaction permettant de démarrer une action.
UIFieldMenuItem	Décrit un élément d'interaction qui permet de naviguer entre les UIUnits, les UISubUnits et les UIgroups.
UIFieldMenu	Représente un ensemble de UIFieldMenuItems qui sont regroupés.

3.3 Le niveau CIM : modèle BPM

3.3.1 Définition du modèle des tâches

Afin de modéliser les tâches, il est important de bien choisir le formalisme ou la notation qui permet de les décrire en prenant en compte l'ensemble des données de l'application (le contenu) ainsi que leurs dynamiques. Comme présenté dans le chapitre 1, il existe plusieurs formalismes permettant la modélisation des tâches. Toutefois, les notations existantes ne permettent pas d'envisager tous les changements de contexte et de prendre en compte la personnalisation ou l'adaptation de l'IHM depuis la phase de conception. Pour cette raison, il est important d'apporter quelques extensions à ces formalismes. Dans la cadre de cette thèse et afin de mettre en œuvre notre modèle des tâches, nous avons adopté la notation BPMN (BPMI, 2004) pour les raisons suivantes :

– C'est l'unique notation qui est standardisée parmi celles présentées dans le chapitre 1 ;
– Elle est plus proche de l'ingénierie des systèmes et, par conséquent, elle permet la modélisation des tâches fonctionnelles et la description de la logique métier de l'application (appelée également le comportement de l'interface (Weyersa *et al.*, 2012)),

– Elle permet de modéliser le flux d'informations inter tâches, ce qui est particulièrement important pour l'intégration de la personnalisation du contenu. BPMN apporte les notions des" messages" et des "flows connexions". Par conséquent, les tâches dans des processus différents peuvent communiquer au moyen des "messages" et celles appartenant à un même processus utilisent les "flow connexion" pour contrôler leurs séquencement.

Néanmoins, il est important de souligner que, dans le cadre de notre approche, BPMN est uniquement utilisée pour exprimer les interactions des utilisateurs tel que proposé par (Brossard *et al.*, 2007) et elle ne sera pas intégrée dans un moteur de workflow. Notre objectif est de l'exploiter dans une approche déclarative afin de générer l'interface utilisateur final, suite à une série de transformations.

En plus des éléments de base fournis par BPMN, nous avons tenu à chercher la bonne manière permettant d'inclure la personnalisation du contenu dans le modèle des tâches. Pour cela, il faut d'abord définir les informations de contexte que nous devrions prendre en compte mais également indiquer les informations du domaine pouvant être influencées par le contexte. Cette correspondance est établie par les mappings précédemment présentés (cf. chapitre 3, §3.2.2).

Enfin, pour mieux définir la façon dont les informations de contexte devrait être utilisées pour fournir des contenus personnalisés, il est important de spécifier, d'une manière abstraite, quels sont les éléments d'interaction à travers lesquels le contenu sera présenté.

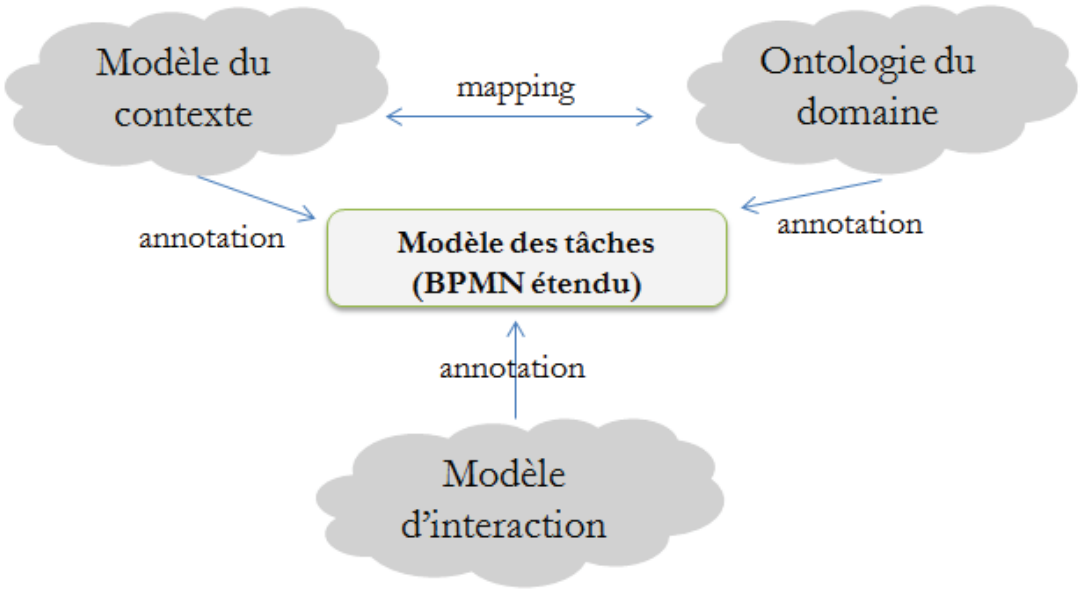


FIGURE 3.9 – Modèle des tâches annoté

En résumé, trois modèles sont utilisés pour soutenir la modélisation des IHM à contenus personnalisés : le modèle de contexte, l'ontologie de domaine et le modèle d'interaction.

L'ensemble des informations issues de ces modèles est utilisé pour annoter le modèle des tâches (cf. Figure 3.9). Dans ce contexte, une extension de BPMN est nécessaire. Après avoir présenté chacun de ces modèles dans les sections précédentes, nous détaillerons l'extension que nous avons apporté à BPMN pour soutenir les annotations au niveau de notre modèle des tâches.

3.3.2 Extension de BPMN pour la modélisation des tâches

Dans le but de prendre en compte la personnalisation du contenu depuis la phase de conception, nous avons étendu la notation BPMN utilisée afin que le concepteur puisse annoter le diagramme avec les éléments issus de l'ensemble des modèles de notre approche. De ce fait, nous avons commencé par une modification des éléments de base fournis par BPMN tels que :

- Les conteneurs qui sont réalisés à l'aide de l'élément BPMN de type *"Pool"*. Dans la version 2.0 de BPMN [1], les conteneurs sont des éléments qui permettent de spécifier un acteur ou une entité organisationnelle particulière. Dans notre modèle des tâches, un *"Pool"* est utilisé comme étant une entité organisationnelle représentant un espace de travail qui regroupe un ensemble d'éléments liés à un même objectif métier. Pour indiquer l'acteur d'une tâche, nous avons fourni la possibilité de le mentionner directement sur la tâche concernée. Le deuxième type de conteneurs proposé par BPMN est l'élement *"Lane"*. Étant donné que nous avons changé le rôle des *"Pools"*, nous n'aurons plus besoin d'utiliser les "*Lanes*" au sein de notre modèle BPMN proposé ;
- Les nœuds du graphe nommé *Contained Element Type* qui représentent les événements (*Start Event, Intermediate Event* et *End Event*), les branchements conditionnels ou les *"gateways" (Exclusive Gateway*, *Inclusive Gateway* et *Parallel Gateway*) ainsi que les tâches. Dans notre modèle des tâches proposé, nous avons défini deux types de tâches. Des tâches réalisées par l'utilisateur appelées *User Task* et des tâches réalisées par le système, appelées *System Task*. Pour des raisons d'organisation, ces nœuds pourraient être regroupés en utilisant, soit l'élément *"SubProcess"*, soit l'élément *"Group"*. L'idée de spécifier les acteurs directement sur les tâches (*User Task* qui est effectuée par un être humain intervenant dans le processus d'interaction et *System Task* qui est effectuée par le système) a été adoptée du travail de (Brossard *et al.*, 2007) ;
- Les arcs qui représentent le flux d'informations. Il y a deux types d'arcs : le *Sequence Flow* pour montrer l'ordre dans lequel les activités seront réalisées dans un processus et le *Message Flow* pour afficher le flux de messages entre les *"Pools"*. Dans notre modèle de tâche, nous autorisons uniquement l'envoi des messages à partir des tâches vers les *"Pools"*. Pour chaque type d'arc, il existe trois sous-types établis grâce à la classe *Condition type* : *Expression* lorsque l'arc est doté d'une condition qui sera

1. http ://www.omg.org/spec/BPMN/2.0/

évaluée lors de l'exécution pour déterminer si le flux concerné sera suivi ou pas ; *None* pour se référer au flux qui ne contient aucune condition ; et *Default* pour les *gateways* exclusives ou inclusives, afin d'indiquer que ce type de flux est celui suivi par défaut.

Une fois que nous avons défini un modèle des tâches à l'aide des ces éléments, nous procédons à son annotation afin d'y intégrer la personnalisation du contenu. Cette annotation est réalisée grâce aux éléments suivants :

- L'attribut *RelatedInteractionElement* qui permet l'association d'un élément d'interaction à une tâche spécifique. En effet, chaque espace de travail contient un certain nombre d'éléments qui le construit et chaque élément sera associé à la tâche qu'elle manipule. Cette association n'est pas arbitraire, et elle est réalisée en respectant les règles que nous avons définies, comme illustré dans le Tableau 3.5 ;
- L'attribut *OntologyElementName*, qui représente le concept ou les concepts de l'ontologie de domaine lié à la tâche ;
- L'attribut *MappingType* qui permet de spécifier la manière dont l'élément de domaine est associé à l'élément du contexte. Dans le cas d'un mapping indirect, nous avons proposé également la classe *Inference Criteria* qui permet de préciser les critères qui seront utilisés lors de la recherche d'une information spécifique.

La Figure 3.10 présente notre modèle des tâches proposé qui permet de mettre en œuvre la personnalisation du contenu (adapté de la version 2.0 de BPMN). Les éléments mis en évidence par la couleur grise sont les parties que nous avons ajoutées au modèle BPMN, afin de parvenir à la personnalisation du contenu.

Table 3.5 – Association entre les éléments BPMN et les éléments d'interaction

Élément BPMN	**Élément d'interaction associé**
Pool	UIGroup
SubProcess	UIUnit
	UISubUnit
	UIFieldMenu
User Task	UIFieldAction
	UIFieldInManual
	UIFieldNavigation
	UIFieldNavigation
	UIFieldInMultiChoices
	UIFieldInOneChoice
System task	UIFieldOut
	UIFieldStatic
Group	UIFieldMenu

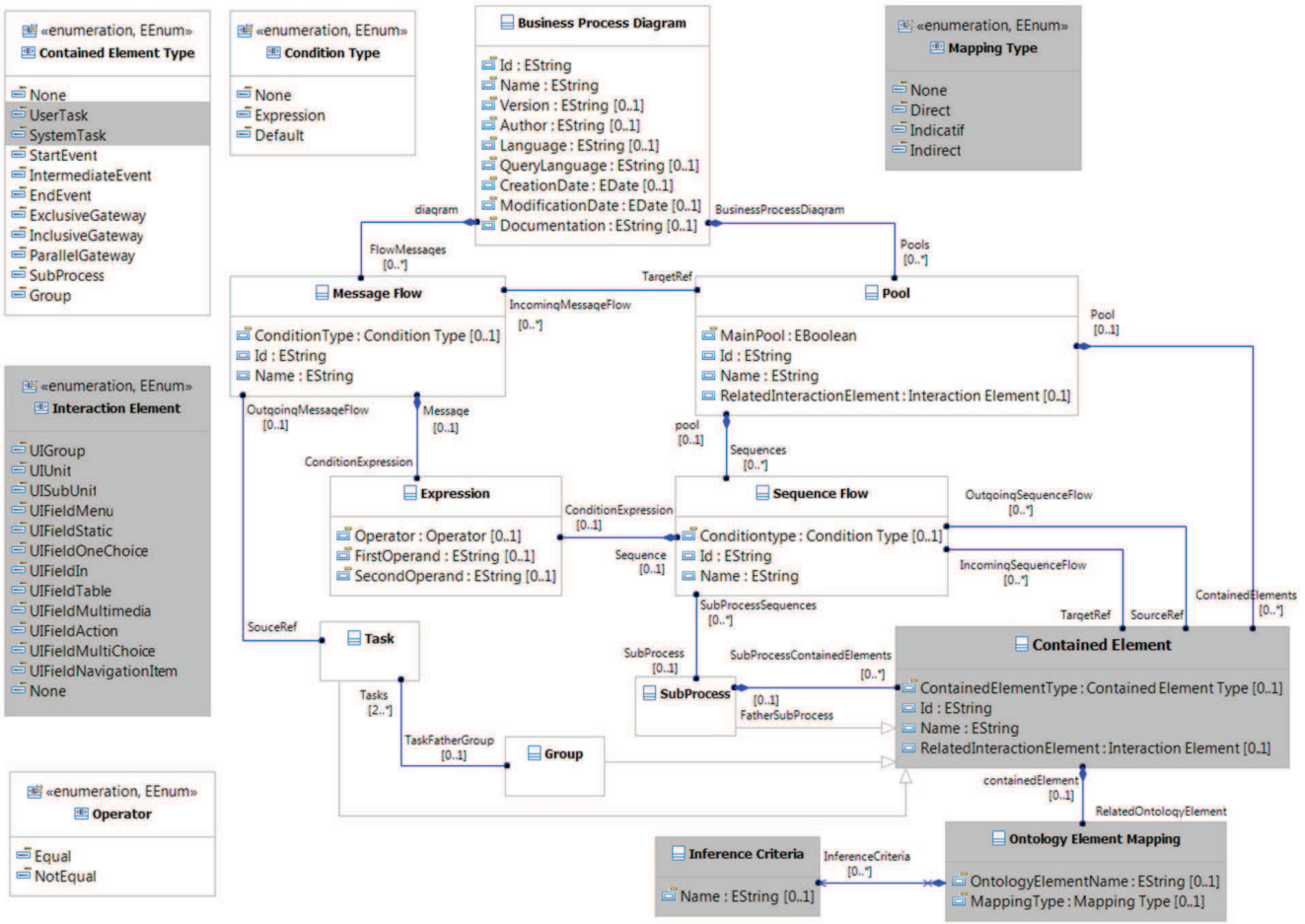


FIGURE 3.10 – Modèle BPMN étendu pour la personnalisation du contenu

3.3.3 Conception des IHM à contenus personnalisés

Dans cette section, nous présentons la manière avec laquelle nous pouvons modéliser une IHM à contenus personnalisés en nous appuyant sur le modèle des tâches. Cette phase de conception s'effectue en deux étapes principales :

1. Élaboration du modèle des tâches non annoté.
2. Annotation du modèle des tâches par les éléments de personnalisation.

3.3.3.1 Élaboration du modèle des tâches

Tout d'abord, le concepteur de l'IHM doit commencer par préciser les conteneurs qui sont mis en place à travers l'élément *"Pool"*. Comme expliqué précédemment, chaque *"Pool"* sera utilisé comme une entité organisationnelle qui correspondra à un espace de travail dans l'interface finale.

Ensuite, le concepteur doit modéliser les activités (*"Task"* et *"Subprocess"*) appartenant à chaque conteneur. Pour chaque tâche, le concepteur doit préciser s'il s'agit d'un *UserTask* ou d'un *SystemTask*.

Afin de synchroniser tous les éléments BPMN crées précédemment, le concepteur doit utiliser les *"events"*, *"gateways"* ainsi que les *"arcs"* en respectant les règles définies par le standard BPMN 2.0.

3.3.3.2 Annotation du modèle des tâches

Après l'élaboration d'un modèle des tâches non annoté, le concepteur annote, si nécessaire, les composants BPMN suivants : *"Pool"*, *"Task"*, *"Subprocess"* et *"Group"*, dans le but d'intégrer la personnalisation du contenu. Nous désignons par la lettre A, chaque élément appartenant à cet ensemble de composants.

Puisque chaque espace de travail contient un ensemble d'éléments qui le construit, chacun de ces éléments sera associé à un composant A. De ce fait, le concepteur peut associer à chaque composant A l'élément d'interaction qui lui est approprié (via l'attribut *RelatedInteractionElement*), tout en respectant les règles définies dans le Tableau 3.5.

Ensuite, en utilisant l'attribut *OntologyElementName*, le concepteur doit associer à chaque tâche d'entrée/Sortie, le concept ou les concepts de l'ontologie de domaine manipulés par cette tâche.

Enfin, lors de l'annotation d'une tâche avec le concept de l'ontologie de domaine, le concepteur doit aussi préciser, le type du mapping (par la classe *Ontology Element Mapping*), conformément aux règles suivantes :

- **Règle 1** : Le mapping direct doit être défini lorsque les éléments de l'ontologie et du contexte ont la même signification sémantique. Ce genre de mapping est toujours utilisé avec les *UserTasks* qui sont associés à l'élément d'interaction *UIFieldInManual*. Si le concepteur prévoit la valeur de ce champ d'entrée d'information, il doit annoter la tâche qui lui est associée, par le concept de l'ontologie de domaine correspondant (par l'attribut *OntologyElementName*). Ensuite, il doit spécifier l'option *Direct* (par l'attribut *mapping type*).
- **Règle 2** : Le mapping indicatif est toujours utilisé dans le cas de sélection d'une information. Par conséquent, il est toujours utilisé avec un *UserTask* qui est associé à un élément d'interaction de type *UIFieldOneChoice* ou bien *UIFieldMultiChoice*. Dans les deux cas, le choix de la meilleure information sera influencé par le contexte. Pour intégrer cet aspect dès la phase de conception, et par analogie avec mapping direct, le concepteur doit associer à la tâche correspondante le concept de l'ontologie de domaine et spécifier qu'il s'agit d'un mapping indicatif. Cela signifie que la sélection ou non d'une telle information est dépendante de la valeur de l'élément de contexte qui lui est mappé.
- **Règle 3** : Le mapping indirect est utilisé lorsque l'information (un concept de l'ontologie) dépend d'un autre. Il est toujours utilisé avec les composants BPMN de type *SystemTask* aux quels sont associés des éléments de type *UIFieldOut*, étant donné que le concepteur estime que la tâche correspondante servira comme une sortie d'informations. Lors de l'annotation des tâches, le concepteur doit préciser le concept de l'ontologie principal (par l'attribut *OntologyElementName*) qui représente l'information à rechercher. Ensuite, il doit indiquer (par la classe *Inference Criteria*) les autres concepts de l'ontologie qui représentent des critères à prendre en compte pendant le processus de la recherche.

  L'ensemble des axiomes et des relations de l'ontologie entre les classes pourraient être exploitées lors du mapping indirect. Du fait que nous nous intéressons à fournir du contenu personnalisé à l'utilisateur, la requête initiale pourrait être étendue en incluant automatiquement d'autres critères lors du processus de recherche. Cette proposition sera détaillée dans le chapitre 4, §4.3.1.2.

Dans le chapitre 5, nous présenterons deux cas d'études permettant d'illustrer et de mettre en œuvre l'ensemble des modèles conceptuels ainsi que notre méthode de modélisation proposé.

3.4 Le niveau PIM : modèle PIIM

Étant donné que le niveau PIM est indépendant de la plateforme, il est important de trouver un modèle permettant de respecter un compromis entre les contraintes suivantes :

a) Pouvoir traduire les aspects du modèle des tâches, du niveau CIM ;

b) Être lié à l'informatisation ;

c) Être suffisamment générique pour garder son indépendance de toute plateforme particulière.

L'approche traditionnelle pour résoudre le problème de description d'une IHM, indépendamment de la plateforme, consiste à écrire différentes versions pour chaque type de plateforme. Cela nécessite le développement de nombreuses versions autant que le nombre de plateformes, ce qui peut induire des problèmes de maintenance et de cohérence. Afin de pallier à ces problèmes, il sera intéressant de faire appel à un format générique, flexible qui permet la description de tout type de plateformes. Pour répondre aux contraintes (a) et (b), nous avons proposé le PIIM (Platform Independent Interaction Model) qui repose sur le langage UIML (cf. chapitre 3, §3.4.1) et afin de répondre à la troisième contrainte (c), nous avons eu recours à un vocabulaire générique (cf. chapitre 3, §3.4.2).

3.4.1 Le modèle PIIM

Afin de définir notre modèle PIIM, nous avons eu recours au langage UIML. Ce langage a retenu notre intérêt étant donné que :

- UIML est un méta-langage. Il définit un ensemble de balises qui sont indépendantes de toute modalité d'interaction, de toute plateforme cible (PC, téléphone, etc.) et de tout langage de développement (Java, C#, etc.). La spécification d'une interface utilisateur se fait à travers un vocabulaire qui spécifie un ensemble de parties qui constituent l'interface ainsi que l'ensemble des propriétés de ces parties.
- UIML sépare les éléments d'une interface utilisateur et identifie l'ensemble des parties composant l'interface et leurs styles de présentation ; le contenu de chaque partie ainsi que sa liaison avec des ressources externes ; et le comportement de chaque partie, exprimé comme un ensemble de règles avec des conditions et des actions.
- UIML permet de grouper les différentes parties de l'interface sous forme d'arbre dont la structure peut changer dynamiquement.
- UIML permet aux différentes parties de l'interface d'être définies dans des modèles (`template`) : ces modèles peuvent être réutilisés, par la suite, dans la modélisation d'autres IHM.
- UIML fournit la possibilité de travailler avec des informations dynamiques et de traduire le passage du flux d'information entre les tâches du modèle CIM. En effet, la notion de variable introduite dans la version 4.0 de UIML peut être exploitable pour ce passage de flux (cf. chapitre 4, §4.3).
- Il existe plusieurs outils permettant la conversion du code UIML vers d'autre langages de programmation (cf. chapitre 4, §4.1.2.3).

Notre modèle PIIM est composé des parties `structure`, `behavior`, `content`, et `style`. Pour manipuler le contenu, UIML offre deux choix : soit de l'intégrer dans la partie `style`, soit de le séparer dans la partie `content`. La seconde alternative est utile uniquement si les concepteurs veulent attribuer plusieurs contenus à un élément d'interaction et seulement si le contenu en sortie est connu d'avance. Pour cette raison, nous avons adopté le premier

choix en y intégrant le contenu dans la partie `style`. Dans le PIIM, la partie `style` contient uniquement les propriétés liées au contenu.

En UIML, la partie `structure` est composée d'un ensemble de parties nommées `parts`. Chaque `part` est dotée d'un ensemble de propriétés qui permettent de la caractériser. Afin que notre modèle PIIM soit indépendant de la plateforme, nous avons eu recours à un vocabulaire générique permettant de spécifier l'ensemble des `parts` ainsi que leurs propriétés d'une manière indépendante de la plateforme. Dans ce qui suit, nous présentons ce vocabulaire.

3.4.2 Vocabulaire générique

Le but du vocabulaire générique est de fournir aux concepteurs, un moyen qui leur permet de créer des IHM indépendamment de toute plateforme et de tout langage de développement. Un vocabulaire générique permet de :

- Fournir un ensemble de widgets génériques communs entre les différentes plateformes. Ces widgets peuvent par la suite être mappés vers d'autres qui sont spécifiques à une plateforme cible. Ces widgets peuvent être divisés en deux types : les conteneurs qui représentent les éléments qui peuvent en contenir d'autres ; les contrôleurs qui représentent les widgets élémentaires. Le Tableau 3.6 définit l'ensemble des widgets génériques dont les pluparts ont été proposés par (Harmonia, 2002) ;

TABLE 3.6 – Définitions des widgets génériques

	Élément	**Définition**
Conteneurs	G :TopContainer	Le conteneur générique qui contient tous les autres composants de l'IHM
	G :Area	Il représente une partie de son conteneur
	G :SplitArea	Il représente une sous-partie du G :Area
	G :Menu	Il décrit tout type de menu
	G :Group	Il regroupe un ensemble d'autres éléments d'une manière logique
	G :List	Il décrit une séquence ordonnée d'autres éléments
Contrôleurs	G :Button	Le contrôle qui représente un seul mode d'interaction
	G :Dialog	Un élément qui permet de contenir un ensemble d'informations à retourner à l'utilisateur
	G :Label	Il contient le label de n'importe quel autre élément d'interaction
	G :MenuItem	Il représente une partie élémentaire du G :Menu
	G :CheckBoxButton	Il permet de choisir ou pas un élément
	G :TextRegion	Il fournit à l'utilisateur la possibilité d'entrer des informations

- Créer un ensemble générique de propriétés pour définir tous les aspects de base de chaque widget générique (ex. la propriété *g :text* permet de spécifier le texte associé à un widget ; *g :selected* permet de vérifier si un widget est sélectionné ou pas ; et *g :enabled* permet de décider si le widget est autorisé pour être affiché...). Pour connaitre la totalité des propriétés disponibles, le lecteur peut se référer au (Harmonia, 2002).

3.5 Le niveau PSM : modèle PSIM

Ce modèle, appelé PSIM, est composé des parties `style`, `presentation` et `logic` de UIML :

– La partie `style` contient les informations de mise en forme utilisant les informations de présentation les plus appropriées en se basant sur la plateforme cible choisie. En effet, elle contient une liste de propriétés et leurs valeurs qui sont utilisées pour décrire l'IHM. Ces propriétés décrivent la façon avec laquelle l'interface sera présentée sur la plateforme cible en terme de polices, couleurs, mise en page, etc. Par exemple, le fragment suivant permet à toutes les `parts` dont la classe est "c1" d'utiliser la police "Comic", et l'élément nommé "n1" aura la taille 100 sur 200.

```
1 <style id="Graphical">
2    <property part-class="c1" name="font" >Comic</property>
3    <property part-name="n1" name="size" >100,200</property>
4 </style>
```

– La partie `peers` dans UIML permet d'associer les éléments génériques de l'IHM ainsi que les méthodes utilisées à leurs homologues dans la plateforme cible. En UIML, toutes les informations concernant la plateforme cible sont stockées dans la partie `peers`. Cette partie est constituée des deux sous-parties `presentation` et `logic`. Généralement, un auteur n'écrit pas la partie `peers`, mais il réutilise ceux qui existent préalablement (Phanouriou, 2000).

– La partie `presentation` sert à effectuer la correspondance entre les éléments UIML génériques et ceux spécifiques à une plateforme cible. Les concepteurs des IHM peuvent créer leurs propres vocabulaire puis l'associer à cette plateforme. Le code UIML suivant exprime le fait que le code générique doit être transformé en utilisant le vocabulaire qui se trouve à l'adresse http ://Java_base.uiml combiné avec celui se trouvant à l'adresse http ://MySwing_#vocab.

```
1 <presentation source="http://MySwing_#vocab"
2               base="http://Java_base"/>
```

– La partie `logic` contient les correspondances entre les méthodes utilisées dans la partie `behavior` et celles qui seront utilisées sur le code source de la plateforme spécifique. Ces correspondances peuvent être implémentées à travers l'appel à des scripts ou des méthodes qui seront invoquées lors de l'interaction de l'utilisateur avec l'IHM finale.

Ainsi, la partie `logic` agit comme une colle entre une IHM décrite en UIML et d'autres codes sources. Le fragment du code qui suit représente un exemple de la partie `logic` indiquant qu'il y a un objet externe dont le nom est "lamih.exemple" et dont l'identifiant dans le code UIML est "Objet". Cet objet possède une méthode externe appelée "Méthode-Plateforme" et dont l'identifiant est "Methode" qui a comme paramètre "param1" de type "int". Ces méthodes seront utilisées par la suite, pendant la recherche d'information, afin de mettre en œuvre la personnalisation du contenu (cf. chapitre 4, §4.3).

```
<logic>
  <d-component id="Objet" maps-to="lamih.exemple">
     <d-method id="Methode" return-type="int" maps-to="Methode-Plateforme">
        <d-param id="param1" type="int"/>
     </d-method>
  </d-component>
</logic>
```

3.6 Processus de génération d'IHM

Après avoir introduit l'architecture globale de notre approche ainsi que les différents modèles qui la définissent, nous décrivons dans cette section les différentes étapes de notre processus de génération des IHM à contenus personnalisés.

Ce processus est une série d'étapes partant d'un ensemble de modèle abstraits vers le code source décrivant l'IHM finale (cf. Figure 3.11). Le modèle des tâches est une notation abstraite qui peut être utilisée dans n'importe quel domaine. Afin qu'elle soit utile pour modéliser les IHM à contenus personnalisés, le concepteur doit *annoter* ce modèle des tâches par les modèles de personnalisation en suivant la manière décrite dans le chapitre 3, §3.3.

Ensuite, et après avoir préparé un modèle des tâches annoté qui est valide, le développeur peut *transformer* ce modèle vers un code UIML générique indépendant de toute plateforme. Ce code UIML généré permet de mettre en œuvre la personnalisation du contenu. Il est composé principalement des parties `structure` et `behavior` ainsi que la partie `style` de UIML, spécifique au contenu.

Mais comme ce code obtenu n'est pas complet, il faut lui rajouter les parties manquantes permettant de lier les éléments génériques à ceux spécifiques à une plateforme cible. Cette étape de passage du code UIML générique vers celui spécifique est appelé une *transition*. Après cela, le développeur peut intervenir au niveau des parties UIML ajoutées pour les modifier et les enrichir en intégrant davantage d'autres propriétés, en fonction de la plateforme cible. Cette tâche fait partie de l'étape *d'adaptation* du style des éléments de l'IHM en fonction de la plateforme cible.

Après avoir obtenu un code UIML complet, dont le style est adapté à une plateforme spécifique et qui permet de mettre en œuvre la personnalisation du contenu, nous pouvons *générer* le code source de l'IHM finale.

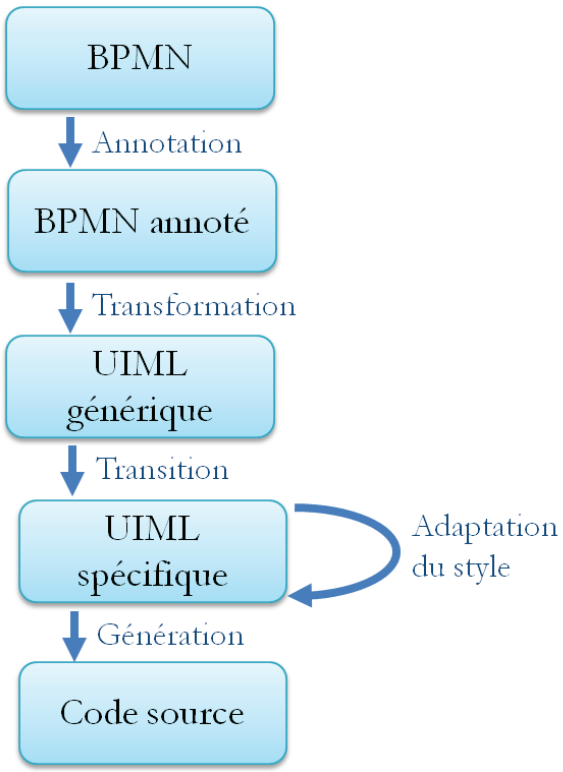


FIGURE 3.11 – Processus de génération des IHM à contenus personnalisés

Dans le cadre de notre approche, le processus de génération des IHM à contenus personnalisés, se résume en six étapes majeures :

1. ***Élaboration*** du modèle des tâches.
2. ***Annotation*** du modèle des tâches par l'ensemble des modèles de personnalisation.
3. ***Transformation*** du modèle des tâches annoté vers du code UIML générique.
4. ***Transition*** du code UIML générique vers du code UIML spécifique à une plateforme.
5. ***Adaptation*** du style (présentation) des éléments de l'IHM en fonction de la plateforme cible.
6. ***Génération*** du code source de l'IHM finale.

Conclusion

Avec la large utilisation des systèmes informatiques pour soutenir nos activités quotidiennes, l'utilisateur devient de plus en plus dépendant de ces systèmes. Afin de rendre ces systèmes plus attractifs, il est important de fournir des systèmes personnalisés pour chacun d'eux. Pour pallier à ces besoins, nous avons apporté les contributions suivantes :

- Proposer une approche de type MDA, permettant de prendre compte la personnalisation du contenu lors de la conception des IHM (Bacha *et al.*, 2011e; Marcal de Oliveira *et al.*, 2013).
- Développer un modèle de contexte qui représente le cœur de cette approche étant donné que la personnalisation du contenu dépend directement du contexte (Bacha *et al.*, 2011e).
- Créer un modèle de mapping qui permet de relier les éléments du modèle de contexte à ceux de l'ontologie de domaine représentant l'ensemble des données de l'application (Bacha *et al.*, 2011e; Marcal de Oliveira *et al.*, 2013).
- Proposer un modèle des tâches développé en se basant sur BPMN. Ce dernier à été étendu et enrichi afin de soutenir la personnalisation du contenu.

Dans le chapitre suivant, nous abordons l'ensemble de ces développements réalisés dans le cadre de cette thèse, permettant de créer, de manipuler et de transformer l'ensemble des modèles conceptuels de l'approche proposée.

Chapitre 4

Vers une instrumentation de l'approche proposée

Introduction

Dans le chapitre précédent, nous avons présenté l'architecture MDA de notre approche ainsi que les modèles supportant ses différents niveaux d'abstraction. Nous avons détaillé, également, les liens entre ces modèles ainsi que la méthodologie à suivre pour concevoir et générer des IHM à contenus personnalisés.

Néanmoins, afin de valider cette approche et de montrer son intérêt fonctionnel, nous avons instrumenté l'ensemble des outils permettant de la mettre en pratique. Pour chaque étape de notre approche, nous avons à choisir les technologies et les outils existants, qui conviennent le mieux à nos besoins pour soutenir une telle étape.

Ce chapitre aborde l'ensemble des développements réalisés dans le cadre de cette thèse, pour créer, manipuler et transformer l'ensemble des modèles conceptuels de notre approche. A cette fin, nous avons eu recours principalement à trois outils qui forment le cœur de l'instrumentation de notre proposition, à savoir EMF, ATL et LiquidApps. Dans ce qui suit, nous commençons par présenter l'architecture globale de cet atelier logiciel ainsi que les différents composants qui la caractérisent. Nous détaillons, ensuite, chacun de ces outils et la manière avec laquelle il est utilisé dans la chaine de génération des IHM à contenus personnalisés.

4.1 Atelier logiciel : de la modélisation à la génération du code

4.1.1 Présentation de l'atelier

Dans le domaine du génie logiciel, un modèle d'architecture n'est intéressant que s'il est associé à une méthode de conception et à des outils de développement (Samaan, 2006). En suivant cette logique et après avoir présenté l'allure globale de notre approche ainsi que les différents liens entre ses modèles, nous essayons désormais d'instrumentaliser notre proposition. Cette mise en pratique est réalisée pour la mise en place d'un atelier logiciel, en se basant sur une chaine d'outils, permettant de générer des IHM à contenus personnalisés.

Cette chaine regroupe principalement trois outils s'exécutant dans un IDE (Integrated Development Environment) Eclipse [1] :

. **EMF (Eclipse Modeling Framework)** : représente un support de modélisation, de métamodélisation et de validation des conceptions. En effet, grâce à EMF nous avons pu mettre en place l'ensemble des modèles de l'approche (BPMN, modèle d'interaction, ontologie de domaine, modèle de contexte, modèle de mapping, UIML). EMF est également utilisé pour instancier l'ensemble de ces modèles afin de les intégrer dans le processus de modélisation et de génération des IHM finales. Cet outil sera utilisé à deux reprises dans le cadre de notre approche.

1. http ://www.eclipse.org/

. **ATL** : ce langage de transformation des modèles est utilisé pour implémenter les règles de transformation, pour pouvoir traduire le modèle BPMN annoté vers le langage UIML.

. **LiquidApps** : cet outil permet d'interpréter le code UIML et de générer un code source vers une plateforme spécifique. A l'issue de cette génération, il devient possible d'intervenir au niveau du code généré pour le modifier, l'enrichir puis le compiler pour générer l'IHM finale. LiquidApps permet la conversion de UIML vers Java, HTML, WML, VoiceXML...

La Figure 4.1 présente l'architecture technique de notre atelier logiciel et la manière avec lesquels les différents outils sont rattachés pour mettre en œuvre la personnalisation du contenu au niveau des IHM générées.

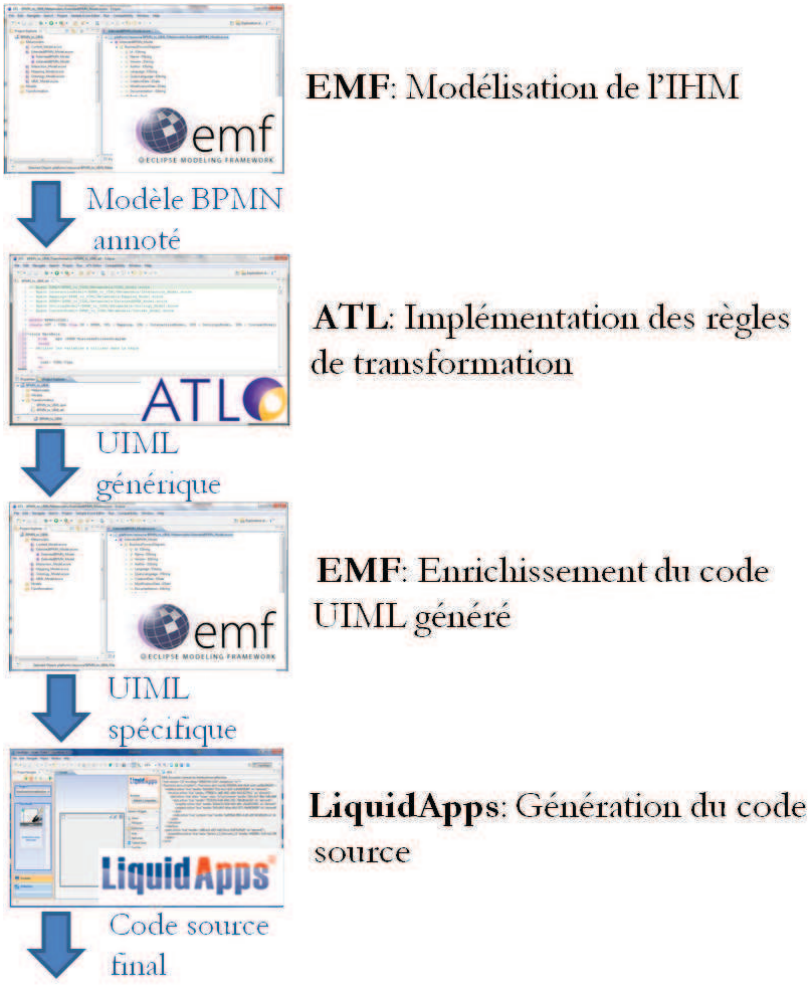


FIGURE 4.1 – Architecture générale de l'atelier logiciel

Le processus de génération d'IHM s'appuie sur 4 étapes :

1. La première étape concerne la création du modèle BPMN annoté réalisé à l'aide de EMF. Elle constitue le point d'entrée pour notre moteur de transformation ATL.
2. La seconde étape exécute les règles de transformations définies dans le chapitre 4, §4.3. Cette étape produit un fichier UIML contenant uniquement les parties `structure` et `behavior` ainsi que la partie `style` spécifique au contenu.
3. La troisième étape cherche à obtenir le code UML complet. Pour ce faire, le concepteur doit intervenir pour enrichir le code UIML généré, à l'aide de EMF, pour y intégrer les parties manquantes spécifiques à la plateforme, à savoir `presentation`, `logic` et le reste de la partie `style`. Le fichier obtenu est toujours un fichier ".uiml" mais qui contient la totalité des balises constituants un document UIML.
4. La dernière étape de ce processus de génération des IHM à contenus personnalisés, est dédiée à l'outil LiquidApps qui permet de traduire le code UIML obtenu vers du code source d'une plateforme spécifique. Ce code peut aussi subir une intervention de la part du développeur afin d'être modifié ou complété.

4.1.2 Composants de l'atelier logiciel : les choix des outils

4.1.2.1 L'outil EMF

✧ **Présentation de EMF**

Eclipse Modeling Framework (EMF) est un environnement qui permet la modélisation, la métamodélisation ainsi que la génération de code au sein de la plateforme Eclipse. Avec EMF, il est possible de définir les modèles de différentes manières. Traditionnellement, les modèles peuvent être construits en utilisant le Java annoté, XML Schema Definition (XSD) ou bien la notation UML de Rational Rose. Indépendamment des formalismes utilisés pour le définir, un modèle EMF est la représentation commune qui les regroupe tous ensemble. En effet, EMF propose également sa propre notation, appelée Ecore (Steinberg *et al.*, 2009), comme langage canonique pour décrire ses modèles. Ainsi, quelle que soit l'annotation retenue pour définir un modèle EMF, nous obtiendrons à la fin un modèle Ecore généré. EMF dispose d'un éditeur d'arborescence proche de XML et d'un éditeur graphique proche de UML.

Un modèle Ecore est, essentiellement, un sous-ensemble du diagramme de classe de UML et peut donc être considéré comme une implémentation du langage MOF proposé par l'OMG. En suivant une approche similaire à MOF, Ecore est défini en utilisant Ecore lui-même, ce qui implique que Ecore est le méta-méta-modèle du méta-modèle Ecore. De cette manière, Ecore permet de définir n'importe quel type de métamodèle. Avec cet objectif, il fournit les formalismes nécessaires pour décrire les concepts et les relations qui les relient. En utilisant Ecore, on peut définir de nouveaux vocabulaires, appelés DSL (Domain Specific Language), qui permettent de travailler avec des modèles dans des contextes différents. Les similitudes entre Ecore et UML sont évidents, étant donné que MOF est basé sur

les diagrammes de classes UML. Cependant, Ecore est le plus préféré par la communauté Eclipse.

Comme indiqué précédemment, un modèle conceptuel peut être décrit en EMF de différentes façons (code Java, XSD, Rational Rose, etc). La représentation canonique d'un modèle est Ecore, alors que son format de persistance est XMI (XML Metadata Interchange) (OMG, 2007) qui est la norme proposée par l'OMG pour l'échange de métadonnées de MOF. Le format XMI a été mis au point afin de permettre la sérialisation de modèles et méta-modèles dans un format physique. Comme son nom l'indique, XMI utilise la syntaxe XML. Pour cette raison, XMI est utilisé pour permettre l'interopérabilité dans l'échange de modèles entre différents formalismes et outils de modélisation (cf. Figure 4.2).

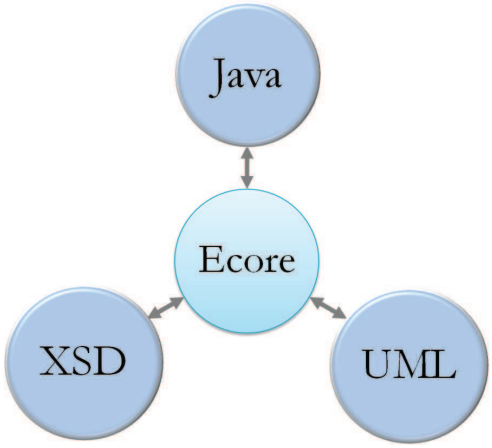


FIGURE 4.2 – Interopérabilité entre Ecore, UML, XSD et Java annoté

✧ Choix de EMF

Après avoir présenté EMF ainsi que ces principales piliers, nous en identifions les raisons pour lesquelles nous avons choisi ce framework pour mettre en œuvre l'ensemble des modèles de notre approche et de le choisir comme un point d'entrée pour notre atelier logiciel :

- La plateforme la plus utilisée dans le domaine de l'IDM est EMF (Louhichi *et al.*, 2011). En fait, il est facile et intuitif à utiliser pour la création, l'édition et la visualisation des modèles. En plus, il fournit la possibilité de valider les modèles créés, une fonctionnalité indispensable pour garder la conformité avec leurs métamodèles.
- EMF fournit la possibilité d'enrichir les métamodèles créés à l'aide des expressions OCL (Object Constraint Language) (OMG, 2012).
- Lorsque nous avons plaidé en faveur de l'utilisation de EMF en tant que framework de modélisation, nous avons pris en compte les avantages prévus EMF en termes d'interopérabilité. En effet, l'extensibilité est indispensable pour tout outil de génie logiciel.

Cette exigence est particulièrement pertinente dans le cadre de l'IDM et dans ce sens EMF permet une migration facile entre le code Java, XSD, UML et Ecore grâce à la possibilité de sérialisation de ces fichiers sous format XMI.

– La génération automatique de code présente l'un des avantages principaux de EMF. En fait, à partir d'un modèle Ecore, il est possible d'obtenir une implémentation Java seulement en quelques clics, en utilisant l'assistant de EMF. Cet atout peut être exploité dans le cadre des perspectives de cette thèse, par le développement d'un éditeur graphique qui permet de définir graphiquement l'ensemble des modèles conceptuels de notre approche, en s'appuyant sur GMF (Graphical Modeling Framework) [2] d'Eclipse. Dans ce cas, EMF permet de générer une partie du code source de l'éditeur.

4.1.2.2 L'outil ATL

✧ **Présentation de ATL**

ATL (ATLAS Transformation Language) est un langage de transformation de modèles accompagné d'une boite à outils (toolkit), initialement proposé par l'équipe AtlanMod [3] du groupe de recherche ATLAS INRIA & LINA (ATLAS-INRIA and LINA, 2006). Le langage est à la fois déclaratif et impératif et permet aux développeurs de transformer un ensemble de modèles d'entrée vers un certain nombre de modèles cibles. ATL dispose des types primitifs (*Numeric*, *String*, *Boolean*), des collections (*set*, *sequence* et *bag*) et d'autres types, qui sont tous des sous-types du type abstrait *OCLAny*. Les opérations de transformation des modèles sous ATL sont appelées *modules* (Jouault et Kurtev, 2005). Ils se composent principalement d'entête *(header)*, de *helpers* et de règles de transformation *(transformation rules)* :

– Le *header* définit le nom du module et les modèles d'entrées/sorties.

– Les *helpers* ressemblent à des fonctions ou des méthodes qui peuvent être appelées dans des règles de transformation ou bien par d'autres *helpers*. Chaque *helper* doit impérativement avoir une valeur de retour.

– Les règles définissent comment les modèles d'entrée sont transformés en modèles cibles. Elles forment l'élément de base dans ATL (Jouault et Kurtev, 2005). Il existe trois types de règles : 1) *Matched rules* qui représentent la partie déclarative de ATL et celles-ci sont automatiquement appelées lors du lancement du processus de transformation. Une règle de type *matched rule* se compose d'un modèle source et un modèle cible ; 2) *Lazy rules* exécutées seulement lorsqu'elles sont appelées par d'autres règles ; et 3) *Called rules*, qui ne comprennent pas d'élément source à transformer mais qui peuvent contenir en option des variables locales ainsi qu'un ensemble de paramètres. La Figure 4.3 montre un exemple d'une *matched rule*.

2. http ://www.eclipse.org/proposals/eclipse-gmf/
3. http ://www.emn.fr/z-info/atlanmod/index.php/Main_Page

```
1  -- Ceci est un commentaire ATL
2  -- En ATL, une transformation s'appelle "module"
3  module Source2Cible ;
4  -- Le mot-clé OUT montre le métamodèle cible nommé "Cible"
5  -- Le mot clé IN montre le méta-modèle source nommé "Source"
6  create OUT : Cible from IN : Source ;
7
8
9  rule Source2Cible {
10 -- L'élément UneClasssSource du méta-modèle source va être traduit vers l'élément
11 -- UneClasseCible du méta-modèle cible
12 from UneClasssSource : Source!SClasse
13 to UneClasseCible : Cible!CClasse(
14 -- L'opérateur "<-" indique que l'attribut "id" de la nouvelle classe va être égal
15 -- au identifiant de la classe source
16 CClasse.id <- SClasse.id
17 )
18 }
```

FIGURE 4.3 – Exemple d'une règle ATL (cas d'une matched rule)

ATL est accompagné d'un ensemble d'outils construits autour de la plateforme Eclipse, appelés ADT (ATL Development Toolkit). Cette boite à outil est composée du moteur ATL de transformation (*Engine block*) et d'un environnement de développement intégré ATL qui comporte plusieurs utilitaires comme les éditeurs dédiés, les débogueurs, la coloration syntaxique, etc.

✧ Choix de ATL

Le choix du langage de transformation représente une tâche importante dans une approche se basant sur l'IDM. Dans le cadre de notre thèse, le choix d'ATL tient à plusieurs raisons :

- Selon l'étude comparative effectuée par (Vara, 2009), comportant une vingtaine de langage de transformation des modèles, le langage ATL est considéré comme étant le langage le plus préféré chez la communauté IDM.
- ATL est mature, très stable et constamment amélioré. En outre, en tant que langage basé sur des règles avec des constructions déclaratives et impératives, ATL établit un bon équilibre entre la facilité d'utilisation et l'expressivité. En effet, ATL est basé sur OCL, par conséquent, il n'est pas difficile pour un développeur ayant une certaine expérience avec OCL de l'apprendre. De plus, il dispose d'une quantité abondante de documentation disponible sous la forme de manuels techniques et de forums pour les utilisateurs.
- L'une des puissances d'ATL est sa capacité à gérer la complexité des transformations de modèles. La manière la plus basique est l'utilisation des *helpers* qui augmente considérablement la lisibilité et la compréhension de la règle de transformation. Elle est souvent utilisée pour itérer et collecter des éléments pour une structure répétitive ou récurrente.
- L'intéropérabilité de ATL avec les outils EMF est une qualité qui a été assurée grâce à sa conformité avec le standard MOF.

– ATL dispose d'un ensemble complet d'outils de développement qui sont construits autour de la plateforme Eclipse fournissant la coloration syntaxique, les rapports d'erreurs, et les éditeurs dédiés (Ajila *et al.*, 2010).

4.1.2.3 L'outil LiquidApps

✧ **Présentation de LiquidApps**

LiquidApps est un outil permettant la gestion du cycle de vie d'un logiciel. Il a été conçue par Harmonia Christopher R. *et al.* (2011) (Harmonia-Group, 2012) en ce basant sur EMF. LiquidApps est doté d'un éditeur de conception des IHM permettant de produire des interfaces indépendantes de toute plateforme spécifique. LiquidApp fut le premier outil qui se base sur UIML pour sérialiser les modèles des IHM créés. Il permet de transformer ce code UIML vers un langage de programmation spécifique.

Dans le cadre de cette thèse, nous n'utilisons qu'une des fonctionnalités de LiquidApps, celle qui consiste à traduire le code UIML vers une plateforme spécifique. La Figure 4.4 présente l'allure générale de LiquidApps. Pour la conception des l'IHM, l'outil dispose d'un éditeur de type WYSIWYG (What You See Is What You Get), doté de la fonctionnalité drug&drop. Une fois l'interface utilisateur spécifiée, le code de l'IHM finale peut être généré automatiquement par LiquidApps vers plusieurs langage cibles tels que Java, C + + Qt, HTML, etc.

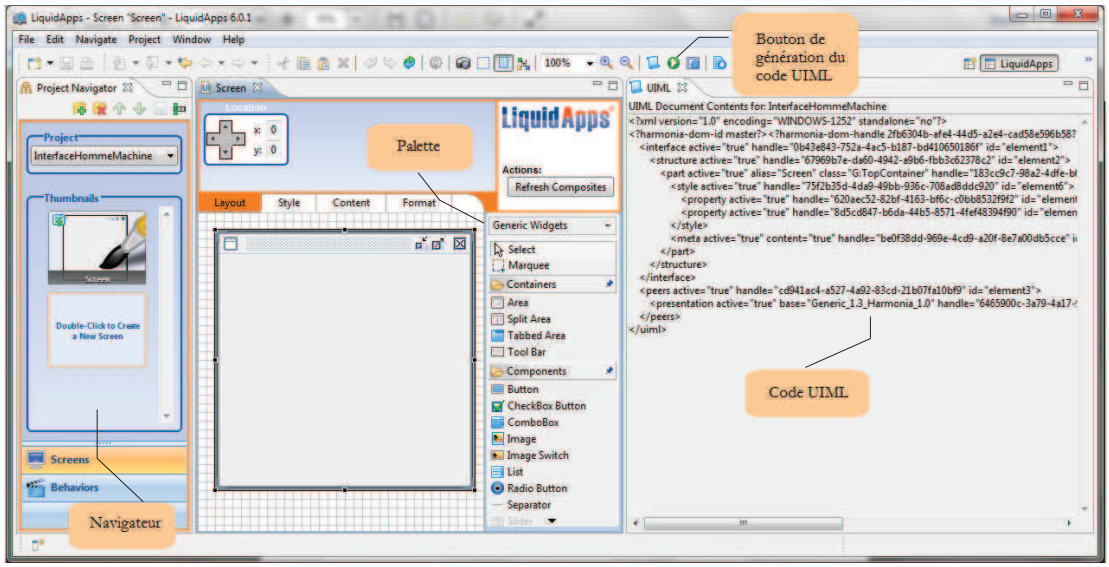


FIGURE 4.4 – Écran principal de LiquidApps

✧ **Choix de LiquidApps**

Dans le cadre de notre approche, la transformation du code UIML vers du code source est notre objectif primaire de la dernière étape. De ce fait, nous devons avoir recours à un moteur de rendu qui traite le code UIML et génère le code source décrivant l'IHM finale. Récemment, une série d'environnements s'inscrivant dans cette lignée, ont été introduits pour permettre la conception d'interfaces utilisateur multi-plateformes. (Luyten et Coninx, 2004), suite à leurs travaux concernant un moteur de rendu de UIML pour la plateforme .NET, ont proposé l'outil UIML.net développé en C#. Cet outil peut rendre un document UIML en utilisant différents ensembles de widgets et de bibliothèques tels Gtk#, System.Windows.Forms, Wx.Net.

Etant donné que l'outil précédemment cité est uniquement spécifique à la plateforme .NET de Microsoft, les mêmes auteurs (Luyten *et al.*, 2006) ont participé à la proposition d'un moteur de rendu pour générer des IHM sur différents types de plateformes. Le moteur de rendu lit le fichier UIML et génère à la sortie l'IHM résultante.

(Meskens *et al.*, 2008) proposent un environnement de création qui prend en charge l'édition graphique des documents UIML 3.0 en offrant une boîte à outils avec un ensemble de widgets prédéfinis. Après avoir sélectionné la plateforme cible, l'outil charge les widgets UIML appropriés, pour cette plateforme.

Jelly (Meskens *et al.*, 2010) a été introduit, récemment, comme étant un outil de conception d'IHM multi-plateformes. Malgré que Jelly repose sur un langage de description d'interface utilisateur propriétaire, il a adopté également la structure du code UIML.

Bien que ces trois dernières propositions visent plusieurs plateformes cibles, elles implémentent leurs outils en se basant sur la version 3.0 de UIML. Dans le cadre de notre thèse, nous avons eu recours à la dernière version de UIML, à savoir la version 4.0, étant donné que nous avons besoin d'exploiter la notion de *variables*, disponible uniquement dans la version 4.0 de UIML.

Dernièrement, (Meixner *et al.*, 2011) ont proposé une version améliorée et étendue de UIML.net conforme à la version 4.0 de UIML. Malheureusement, l'outil proposé n'est pas publique et son usage est réservé uniquement pour le DFKI (Deutsches Forschungszentrum für Künstliche Intelligenz) [4].

4.2 Mise en œuvre des modèles de l'approche

4.2.1 Modèles de conception des IHM

Comme il a été présenté dans le cadre de notre approche dans le chapitre 3, la première étape dans la modélisation des IHM consiste à élaborer un modèle des tâches. Afin de soutenir la modélisation des IHM à contenus personnalisés, ce modèle des tâches est défini

4. http ://www.dfki.de/web

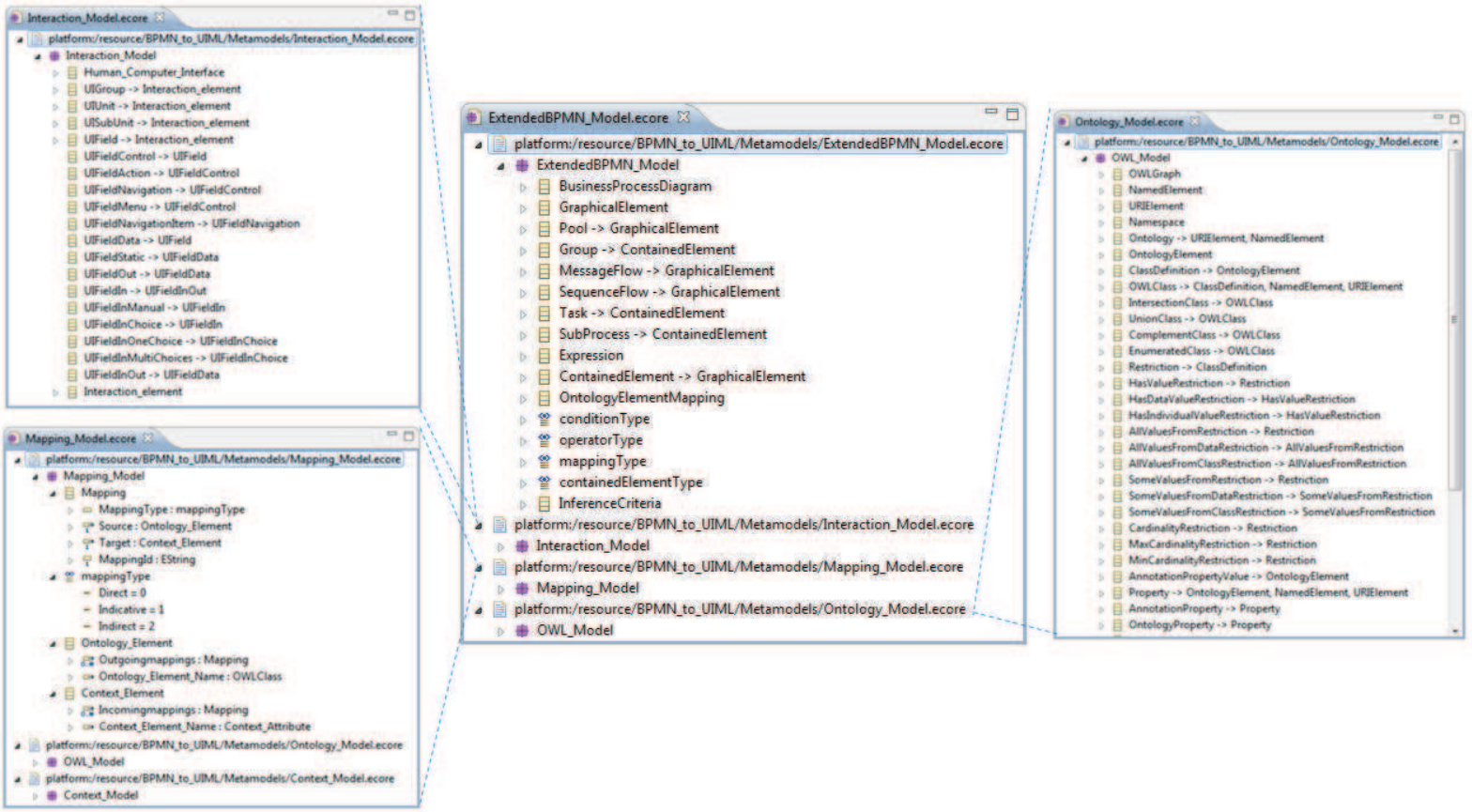


FIGURE 4.5 – Le métamodèle du BPMN annoté, développé en Ecore et édité avec EMF

en BPMN. Il doit tenir compte des modèles de personnalisation, à savoir le modèle de contexte, l'ontologie de domaine, le modèle de mapping et le modèle d'interaction. L'ensemble des informations issues de ces modèles sont utilisées pour annoter le modèle des tâches.

Pour ce faire, nous avons établi explicitement les différentes relations entre ces modèles, en développant, en Ecore, le métamodèle définissant notre modèle de BPMN annoté ainsi que les métamodèles de l'ensemble des modèles de personnalisation. Notre métamodèle des tâches représente d'une manière uniforme les classes, leurs relations ainsi que les restrictions qu'elles subissent. En plus des restrictions définies au niveau du métamodèle de BPMN standard (BPMI, 2004), notre métamodèle de BPMN annoté doit respecter l'ensemble des extensions que nous avons apportées à ce métamodèle et présentées dans le chapitre 3, §3.3. Nous tenons à préciser que pour décrire nos ontologies du domaine, nous avons eu recours au métamodèle de OWL 2.0 proposé par le W3C (World Wide Web Consortium) [5].

La Figure 4.5 présente notre métamodèle BPMN défini à l'aide de la notation Ecore sous forme d'arbre, en tenant en compte les modèles de personnalisation. L'annotation par des éléments de ces métamodèles se fait à travers la fonctionnalité *"Load Ressource"* que fournit EMF. Comme le présente la Figure 4.5, l'ensemble des métamodèles utilisés pour annoter le métamodèle de BPMN apparaît au dessus de celui-ci, a savoir le métamodèle, d'interaction, le métamodèle de l'ontologie et le métamodèle de mapping. Ce dernier fait appel, également, au métamodèle de l'ontologie et à celui du contexte (cf. Figure 4.6).

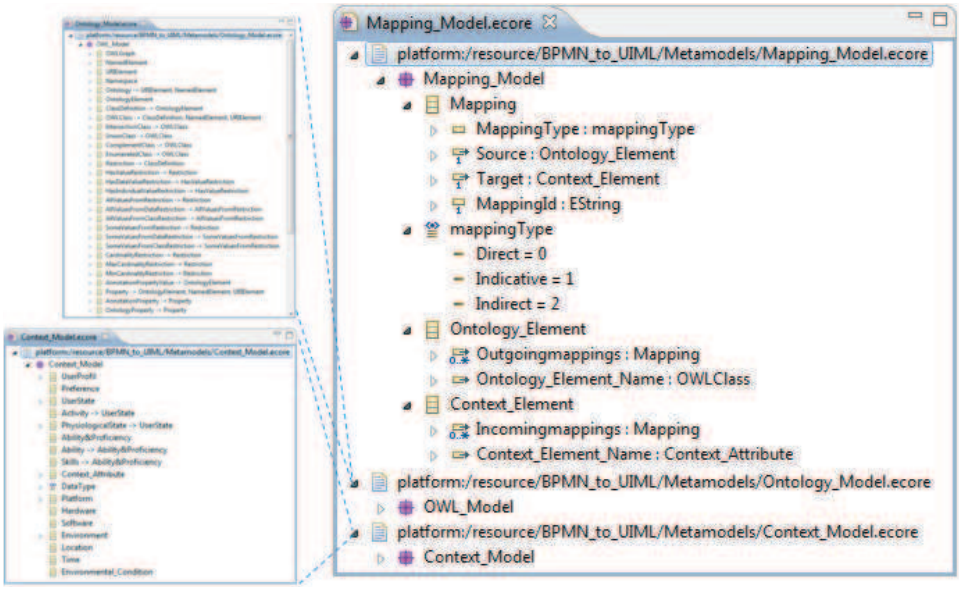


FIGURE 4.6 – Le métamodèle du mapping, développé en Ecore et édité avec EMF

5. http ://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel

Une fois que nous avons un modèle Ecore, nous pouvons utiliser des outils standards de EMF pour créer et manipuler les instances. L'instanciation de notre métamodèle des tâches annoté, produira un fichier nommé *ExtendedBPMN_Model.xmi* qui sera le point d'entrée pour l'outil ATL. Des exemples d'instanciation de ce métamodèle seront présentés dans le chapitre 5.

4.2.2 Modèle de description des IHM

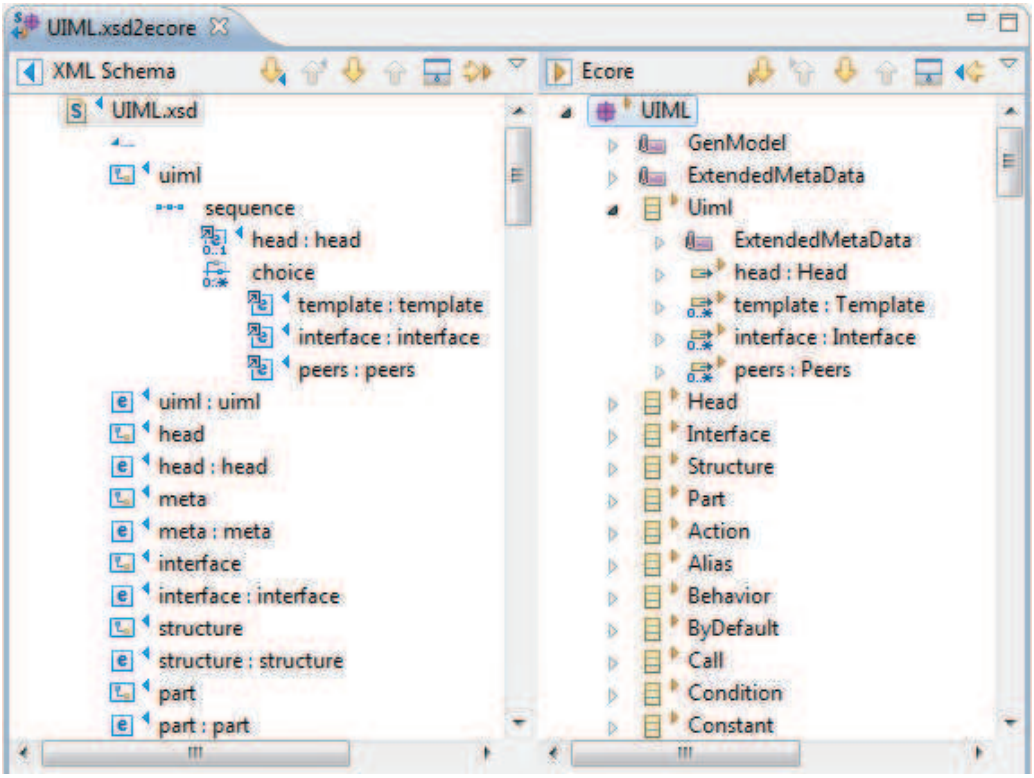


FIGURE 4.7 – Fichier XSD de UIML 4.0 et son métamodèle en Ecore

Après avoir défini l'ensemble des métamodèles de conception des IHM, appartenant au niveau CIM, nous devons également proposer un métamodèle décrivant la cible de notre transformation, à savoir le langage UIML. UIML se base sur le langage XML dont la structure peut être spécifiée via une description DTD. Étant donné que le modèle Ecore de la version 4.0 de UIML n'est pas disponible, nous avons eu recours à la DTD (Document Type Definition) de UIML 4.0, fourni par OASIS (Organization for the Advancement of Structured Information Standards) [6]. Cette DTD définit la structure du document UIML.

6. http ://docs.oasis-open.org/uiml/v4.0/cd01/uiml-4.0.dtd

Par la suite, nous avons converti le fichier uiml-4.0.dtd sous format XSD à l'aide l'outil XMLSpy [7]. En effet, le fichier XSD peut être utilisé pour générer automatiquement un métamodèle Ecore. Ce métamodèle décrit la structure que les modèles UIML doivent respecter. La Figure 4.7 présente le fichier initial XSD de UIML 4.0 ainsi que le métamodèle Ecore automatiquement obtenu. Ce dernier sera utile sur deux plans :

– Comme UIML est la cible de notre transformation, il est nécessaire que nous disposions d'un métamodèle spécifiant ce langage.

– Étant donné que le code UIML généré sera incomplet, il devient plus facile d'intégrer les parties manquantes via l'éditeur de EMF.

L'ensemble des métamodèles proposés dans le cadre de cette thèse, exprimés en Ecore, seront présentés dans l'annexe A2.

4.3 Spécification et implémentation des règles de transformation

Les transformations de modèles représentent la base de toute proposition en IDM étant donné que chaque étape du processus de développement implique une transformation de modèles pour créer ou générer d'autres. De cette façon, une fois que l'ensemble des métamodèles a été défini et les outils permettant leurs transformations mis au point, l'étape suivante consiste à les relier par le biais de transformations de modèles (OMG, 2003). À cette fin, nous devons spécifier et mettre en œuvre les règles de correspondance qui composent chaque transformation de modèles.

(OMG, 2003) affirme que les correspondances entre les modèles sources et les modèles cibles peuvent être définies en langage naturel en utilisant des algorithmes, ou bien en ayant recours à des langages spécifiques qui permettent de mettre en œuvre ces correspondances au niveau conceptuel. Une fois que les règles de correspondance ont été formellement spécifiées, celles-ci peuvent être implémentées pour traduire la spécification formelle en une abstraction opérationnelle. À cette fin, comme nous l'avons soutenu dans le chapitre 4, §4.1.2.2, nous utilisons ATL..

Dans ce qui suit, nous présenterons les spécifications que nous avons mis en place afin de traduire les modèles de BPMN annoté vers du code UIML. Chaque spécification sera argumentée et suivie par le code ATL qui permet de la mettre en œuvre. La Figure 4.8 représente la fenêtre de configuration de ATL pour l'initialisation de la transformation dont le module est désigné par *BPMN_to_UIML.atl*. Les champs IN contiennent l'ensemble des modèles qui subiront la transformation. Ces modèles, exprimés sous format XMI, ne sont que des instances des métamodèles que nous avons développés. Le champs OUT représente le modèle UIML de sortie.

7. http ://www.altova.com/xmlspy.html

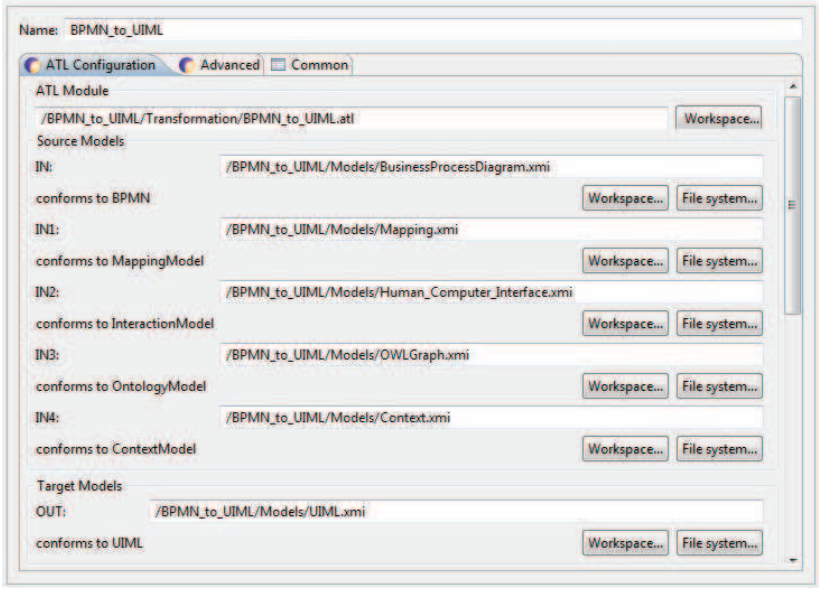


FIGURE 4.8 – Initialisation d'une transformation avec ATL

4.3.1 Du CIM vers PIM

Pour chaque élément du modèle BPM, nous générons une ou plusieurs règles UIML (`rule`), en fonction de sa nature. Dans notre code source ATL, nous avons défini une règle ATL de type "*Matched Rule*" qui représente le programme principal. Elle fait appel à d'autres règles de type "*Called Rule*" ainsi qu'à des "*Helpers*". La Figure 4.9 présente un exemple du code ATL de type "*Matched Rule*" permettant d'initialiser l'ensemble des parties du code UIML généré. Après, en se basant sur l'ensemble des informations venant du BPM, du modèle de contexte, du modèle de domaine et de modèle de mapping, les parties UIML (`structure`, `behavior` et `style`) seront remplies. La manière avec laquelle nous générons chaque partie sera décrite dans les sections qui suivent.

4.3.1.1 Génération de la partie `Structure`

Pour générer la partie `structure`, la transformation dépend de l'élément BPMN et de l'élément d'interaction qui lui est associé. Dans notre code ATL, nous commençons par traiter chaque *Pool* à part, parcourir tous les éléments qui y sont inclus et puis exécuter le traitement approprié. Pour chaque élément d'interaction, une ≺`class`≻ UIML est créée sous la balise ≺`part`≻ en respectant la hiérarchie des éléments. Le Tableau 4.1 introduit l'ensemble de classes UIML qui sont générées pendant les transformations en se basant sur l'élément BPMN et l'élément d'interaction qui lui est associé.

TABLE 4.1 – Classes UIML générées pour la partie `Structure`

Source		**Cible**	
Élément BPMN	**Élément d'Interaction**	**Pré-condition**	**Élément UIML générique créé**
Pool	UIGroup	–	G :TopContainer
SubProcess	UIUnit	–	G :Area
SubProcess	UISubUnit	–	G :SplitArea
SubProcess	UIFieldMenu	–	G :Menu
UserTask	UIFieldAction	–	G :Button
UserTask	UIFieldInManual	–	G :Text
UserTask	UIFieldNavigation	L'élément contenant cet élément est de type G :Menu	G :MenuItem
UserTask	UIFieldNavigation	L'élément contenant cet élément est différent de type G :Menu	G :Dialog
UserTask	UIFieldInMultiChoices	–	G :List
UserTask	UIFieldInOneChoice	–	G :CheckBoxButton
System Task	UIFieldOut	–	G :Area
System Task	UIFieldStatic	–	G :Label
Group	UIFieldMenu	–	G :Menu
Group	–	–	G :Group

```
1  module BPMNtoUIML;
2  create OUT : UIML from IN : BPMN, IN1 : Mapping, IN2 : InteractionModel, IN3 :
       OntologyModel, IN4 : ContextModel;
3  rule MainRule {
4      from    bpd :BPMN!BusinessProcessDiagram
5      using{
6  -- déclarer les variables à utiliser dans la règle
7     } to
8       uiml: UIML!Uiml
9       do {
10 -- Initialiser les différentes parties du code UIML
11         interface <- thisModule.createInterface('Interface' + bpd.Id, uiml);
12         structure <- thisModule.createStructure('Structure' + bpd.Id, interface);
13         style <- thisModule.createStyle('Style' + bpd.Id, interface);
14         behavior <- thisModule.createBehavior('Behavior' + bpd.Id, interface);
15         peers <- thisModule.createPeers('Peers' + bpd.Id, uiml);
16         logic <- thisModule.createPeersLogic(bpd.Id, peers);
17         presentation <- thisModule.createPeersPresentation(bpd.Id, peers);
18 -- Suite du code ATL permettant le remplissage des parties Structure, Behavior
19 -- et Style (celle qui concerne le contenu)
20       }}
```

FIGURE 4.9 – Règle ATL permettant l'initialisation des différentes parties du code UIML

La Figure 4.10 présente une partie du code ATL qui génère la partie `structure` du PIIM liée à l'élément *Pool*. Pour chaque élément de type *Pool* dans le modèle BPM, un élément ≺`part`≻ est créé avec l'attribut ≺`class`≻ de type `G:TopContainer`. Ensuite, nous analysons tous les éléments qui composent ce *Pool*. Pour chaque élément, un ≺`part`≻ est créé avec l'attribut de la classe correspondante, et avec le même identifiant (*id*) que celui de l'élément du modèle BPM.

```
1  rule UIMLStructureFromPool {
2     using
3     partpool: UIML!Part ()= OclUndefined; }
4     do {
5     if  (pool.RelatedStaticElement.oclIsTypeOf(InteractionModel!UIGroup)
6       {partpool <- thisModule.createStructurePart(pool,'G:TopContainer', structure);
7          if (pool.Related_Static_Element.oclIsTypeOf(InteractionModel!UIGroup))
8  -- traitement de des "Subrocesses" contenus dans le Pool
9             {for ( x in pool.ContainedElements)
10                {if ( x.oclIsTypeOf(BPMN!SubProcess))
11               thisModule.SubProcess(x,partpool);
12 -- traitement de des "Tasks" contenus dans le Pool
13               if ( x.oclIsTypeOf(BPMN!Task))
14               thisModule.Task(x,partpool);
15 -- traitement de des "Groups" contenus dans le Pool
16               if ( x.oclIsTypeOf(BPMN!Group))
17               thisModule.Group(x,partpool);}}}}}
18
19 -- règle appelée dans le code précédent permettant de créer un "part" sous la
       partie
20 -- Structure
21
22 rule createStructurePart(pool: BPMN!Pool, class: String, structure: UIML!Structure)
       {
23     to  p: UIML!Part ()
24     do {
25     p.class <- class;
26     p.id <- pool.Id;
27     structure.part <- p;
28     p;
29       } }
```

FIGURE 4.10 – Exemple d'une règle de transformation en ATL générant la partie `structure` à partir de l'élément BPMN *Pool*

4.3.1.2 Génération de la partie `Behavior`

Un modèle de tâche de type BPMN modélise les processus contenant des tâches interactives et non interactives. Pendant le passage de BPMN vers UIML, il est impératif de garder la même logique et de traduire toutes les tâches définies.

Afin de décrire le comportement de l'application pendant l'interaction avec l'utilisateur, UIML fournit la partie `behavior`, définie au moyen d'un ensemble de règles (`Rule`), dont chacune se compose en outre d'une `condition` et d'une `action`. Dans le but de garantir le séquencement des différentes tâches, UIML dispose de la balise ≺`Event`≻ qui pourrait

être une condition pour déclencher une règle UIML ou bien qui pourrait être le résultat d'une action d'une règle. Cette balise ne permet de représenter que les tâches interactives produites par l'utilisateur ou le système. Or, nous devons aussi traduire les tâches non interactives.

La solution proposée consistait à créer la notion de *variable d'activation* à fin d'assurer et de conserver la dynamique et le passage du flux d'information entre tâches interactives et tâches non interactives lors du passage du BPMN vers UIML. En effet, pour chaque élément du diagramme BPMN, nous générons au niveau du code UIML une variable *booléenne* appelée *variable d'activation* et dont le nom est de la forme `Elementid + Isactivated`. Dès que la *variable d'activation* d'un élément est égale à `true`, une suite d'actions UIML est générée selon le type de l'élément BPMN. Dès leurs créations au niveau du code UIML, toutes les variables d'activation vont être initialisées à `false` sauf celles spécifiques à l'élément BPMN de type *Pool* initial (celui qui contient le premier processus) étant donné qu'elles représentent le déclencheur du processus (Bacha *et al.*, 2011d).

Nous avons également créé un deuxième type de variable *booléenne* spécifique aux éléments BPMN de type *gateway exclusive*, appelée *variable de validation*. En effet, pour chaque *gateway*, nous créons dans le code UIML généré une variable qui prend la forme suivante `VariableGateway + Elementid + isValidated`. Cette variable servira à assurer le suivi d'un seul chemin sortant d'un *gateway*. Dès leurs créations au niveau du code UIML, toutes les *variables de validation* vont être initialisées à `false` et dès qu'une branche sortant d'un gateway est suivie, cette variable aura la valeur `true`.

Comme mentionné précédemment, nous cherchons à mettre en œuvre deux méthodes de personnalisation, à savoir le remplissage automatique des formulaires et l'enrichissement des requêtes :

- ***Le remplissage automatique des formulaires*** est réalisé pour tout champ doté d'un mapping direct ou indicatif au niveau CIM pendant la conception de l'IHM. À cette fin, nous tenons compte du concept de l'ontologie et de l'élément du contexte qui lui est mappé. Pour les mappings directs, dans la partie `when-true,` la propriété *g :text* sera remplie automatiquement par la valeur prise du contexte pendant l'exécution en appelant la méthode "*GetValueFromContext*" qui prend comme paramètre l'élément de l'ontologie lié à la tâche. Dans le cas des mappings indicatifs, la valeur du contexte est vérifiée dans la partie `condition` et en fonction de cette valeur, la partie `when-true` agit sur la propriété *g :selected* en la mettant à `true` ou `false`. Dans la partie `behavior`, le code UIML généré manipule les propriétés de `style` liées au contenu (tels que *g :text*, g :*selected*...) ;
- ***L'enrichissement de la requête de recherche*** est réalisé via le mapping indirect. En effet, lors de la transformation des modèles de niveau CIM, nous générons sous la partie `when-true` une balise UIML de type ≺`call`≻. Cette balise ≺`call`≻ représente un appel à une méthode abstraite ou un service externe (qui utilise un langage autre que UIML) qui sera codé ultérieurement par le développeur afin de rechercher les informations nécessaires en se basant sur des paramètres donnés. Cette

méthode est équivalente à une requête de recherche où l'attribut *DomainElementName* présente l'élément recherché, le premier argument de la méthode, et le *inferenceCiteria* présente les paramètres à prendre en compte lors de la recherche. Néanmoins, puisque nous sommes intéressés par fournir un contenu personnalisé à l'utilisateur, nous décidons d'enrichir cette requête en incluant automatiquement d'autres éléments du contexte pendant le processus de la recherche. À cette fin, nous utilisons les relations et les axiomes de l'ontologie.

L'idée est d'exploiter les classes de l'ontologie qui sont dotées d'un mapping indirect et que le concepteur n'a pas explicitement spécifiée pendant la conception en tant que critères d'inférence. En supposant que le *DomainElementName* soit nommé "A" et que "B" appartienne à son *inferenceCiteria*, chaque classe de l'ontologie qui dispose d'un mapping indirect et qui est sur le chemin de "A" à "B" sera intégrée au sein des *inferenceCiteria*, et par conséquent elle va enrichir la requête (Marcal de Oliveira *et al.*, 2013). De ce fait, nous avons besoin de trouver un chemin orienté allant de "A" vers "B" et qui contient des classes pertinentes qui permettent d'enrichir la requête. On dit qu'une classe "C" est pertinente lorsque :

– Il y a un chemin entre "A" et "B" avec un ensemble de relations "R"
– "C" est une classe utilisée sur ce chemin
– "C" est dotée d'un mapping indirect
– "C" est différente de "B"

Pour trouver le bon chemin, nous vérifions l'orientation du chemin en utilisant les axiomes de l'ensemble des relations de l'ontologie, comme illustré dans le Tableau 4.2. La Figure 4.11 présente le code de la règle ATL permettant d'identifier l'ensemble des classes construisant le chemin entre deux classes de l'ontologie. Cette règle retourne un ensemble de classes sur lesquelles nous allons exploiter les axiomes de l'ontologie pour enrichir la requête initiale.

TABLE 4.2 – Axiomes de l'ontologie utilisés pour la recherche d'un chemin entre deux classes

Caractéristique de la relation	**Définition**	**Comment exploiter l'axiome ?**
FunctionalProperty	Si une propriété P est marquée comme Fonctionnelle (FunctionalProperty) alors pour toutes classes C, C' et C" : P (C, C') et P (C, C") implique C' = C"	En allant de A à B, on ne peut aller qu'à partir d'une classe intermédiaire C à une autre C'. Il n'est pas permis de passer de C' à C. Il s'agit d'une propriété orientée. Si son orientation n'est pas la même que celle de A vers B, le chemin (A-B) ne sera pas utile pour enrichir les requêtes. Par conséquent, le chemin complet est abandonné.
SymmetricProperty	Si une propriété P est marquée comme Symétrique (SymmetricProperty), alors pour toutes classes C et C' : P (C, C') = P (C', C)	En cherchant un chemin, le passage de C à C' est symétrique. Le passage d'une classe à l'autre est toujours permis dans les deux sens ; i.e. on peut aller de C à C' et vice-versa
TransitiveProperty	Si on définit une propriété P comme Transitive (TransitiveProperty), cela signifie que si un couple (C, C') est une instance de P, et le couple (C', C") est aussi instance de P, alors le couple (C, C") est également une instance de P.	En supposant que "B" est un *inferenceCiteria* pour la recherche de "A" et "X" est une super-classe de "B" qui possède une relation symétrique et transitive à la fois. Chaque classe de l'ontologie qui est dotée d'un mapping indirect et qui est une sous-classe de "X", sera intégrée au sein des *inferenceCiteria* de "A"

```
1  rule ExplorePath(SearchedPath : Sequence (String), Source: String, Target: String,
       Depth: Integer )
2  {
3      using{
4           Taboo : Sequence (Boolean) = Sequence {};  }
5      do   {
6  -- la classe de départ sera le premier élément de la liste
7      SearchedPath <- SearchedPath -> insertAt(Depth,Source);
8  -- on est sur le sommet d'arrivé -> fini: retourner la liste des classes
9  -- se trouvant sur le chemin entre la classe de départ et celle d'arrivé
10     if (Source=Target)
11     {SearchedPath;}
12 -- sinon on explore les chemins restants
13     else
14     {for  ( i in OntologyModel!OWLClass.allInstancesFrom('IN4'))
15        {for (y in OntologyModel!ObjectProperty.allInstancesFrom('IN4'))
16 -- pour chaque classe X qui est "suivante" à la classe source, on vérifie
17 -- que celle-ci n'appartient pas déjà à la liste et qu'elle est différente de la
18 -- classe cible.
19        {if ( y.domain.localName = Source and i.Exist() and y.range.localName = i.
              localName  and SearchedPath->includes(i.localName)=false and
                        i.localName <> Target  )
20 -- si la condition précédente est vrai, alors considérer X comme étant une
21 -- source et relancer la recherche à partir d'elle(Exploration récursive)
22          {SearchedPath<-thisModule.ExplorePath(SearchedPath, i.localName, Target,
                Depth +1);}
23         else
24          {SearchedPath; }  }  }  }  }  }
25
26
27 -- Helper qui permet de vérifer si la classe de l'ontologie possède ou non des
28 -- relations dont elle est la cible
29 helper context OntologyModel!OWLClass def: Exist(): Boolean = if self.
       domainProperty <> Sequence {}
30 then true
31 else false
32 endif;
```

FIGURE 4.11 – Règle ATL permettant d'identifier le chemin entre deux classes de l'ontologie

Le Tableau 4.3, introduit la manière avec laquelle nous remplissons la partie `behavior`. En effet, pour chaque élément de notre modèle BPMN annoté, nous générons des règles UIML (`rule`). Pendant cette génération, nous tenons en compte de l'élément BPMN, de l'élément du modèle d'interaction qui lui est rattaché ainsi que le mapping défini. Ce tableau est constitué de sept colonnes :

- **Colonne 1 :** Présente l'élément BPMN concerné selon lequel une règle UIML (`rule`) va être générée. Cette colonne est nommée "BPMN".
- **Colonne 2 :** Contient l'élément du modèle d'interaction associé à l'élément BPMN (cf. Tableau 4.1). Cette colonne possède comme intitulé "Inter".
- **Colonne 3 :** Présente le type de mapping associé à l'élément BPMN où un mapping peut être Direct, Indicative ou bien Indirect. Cette colonne est nommée "Mapp"
- **Colonne 4 :** Comprend la contrainte à vérifier pour générer la partie `condition`. Cette

colonne est nommée “Contrainte `condition`”. Les contraintes à vérifier sont les suivantes :

. Cnt.C.1 : Sans interaction entre l'utilisateur et cet élément.

. Cnt.C.2 : Avec interaction entre l'utilisateur et cet élément.

. Cnt.C.3 : Si cet élément est contenu dans un *SubProcess.*

. Cnt.C.4 : Si cet élément est contenu dans un *Pool.*

. Cnt.C.5 : Si X est un *MainPool.*

. Cnt.C.6 : Si X n'est pas un *MainPool.*

. Cnt.C.7 : Si la source est un *gateway exclusive.*

. Cnt.C.8 : Si la source est un *gateway inclusive.*

. Cnt.C.9 : Si la source est differente des *gateway exclusive* et *gateway inclusive.*

– **Colonne 5 :** Elle présente le contenu que doit comprendre la partie `condition` dans le code UIML généré. Cette colonne est intitulée “Contenu `condition`” et elle peut avoir cinq type des conditions à vérifier :

. C.C.1 : Vérifier si la variable d'activation est égale à `True`.

. C.C.2 : Vérifier si la variable d'activation est égale à `True` et que cet élément a subit une action par l'utilisateur.

. C.C.3 : Vérifier si la variable d'activation est égale à `True` (Pour le cas des *Pool* autres que le *MainPool*, la variable d'activation de cet élement est initialisée à `False`) et vérifier que l'élément d'interaction qui lui est associé, reçoit un évènement de type `g:actionPerformed`

. C.C.4 : Vérifier si la variable d'activation du *sequence* est égale à `True`, vérifier si la variable d'activation du *gateway* est égale à `True` et vérifier si la condition au niveau de la sequence est vérifiée.

. C.C.5 : Vérifier si la variable d'activation du *sequence* est égale à `True` et vérifier si la condition au niveau de la sequence est vérifiée.

– **Colonne 6 :** Elle indique la manière avec laquelle le code UIML généré peut agir sur le contenu. Cette colonne sera nommée “Contenu”. Notre approche propose huit manières pour agir sur le contenu :

. C. 1 : La propriété `g:text` de l'élément d'interaction va être remplie par la valeur de l'élément de contexte mappé directement avec l'élément de l'ontologie.

. C. 2 : Enregistrer la nouvelle valeur entrée par l'utilisateur.

. C. 3 : La propriété `g:selected` de l'élément va recevoir la valeur `True`.

. C. 4 : La propriété `g:selectedvalue` de l'élément d'interaction va recevoir la valeur de l'élément correspondant.

. C. 5 : Changer le label de l'élément en associant à la propriété `g:text` de l'élément, le texte inscrit sur le *UIFieldAction* qui lui est associé.

. C. 6 : Changer le label de l'élément en associant à la propriété `g:text` de l'élément, le texte inscrit sur le *UIFieldStatic* qui lui est associé.

. C. 7 : Retourner les résultats de la requête permettant de rechercher un élément en fonction des paramètres passés en arguments.

. C. 8 : Changer le titre de l'élément en associant à la propriété `g:title` de l'élément, le nom du *UIGroup* qui lui est associé

– **Colonne 7 :** Elle présente le contenu de la partie `action` dans le code UIML généré. Cette colonne possède comme alias "Contenu `action`" et elle peut contenir huit types d'actions :

. C.A. 1 : Mettre la propriété `g:visible` de l'élément à la valeur `True` et activer les variables d'activation des éléments de type *sequence* ou *message* qui partent de cette tâche.

. C.A. 2 : Mettre la propriété `g:visible` de cet élément à la valeur `True` et mettre à la valeur `True` les variables d'activation des éléments de type *sequence* qui partent de cette tâche.

. C.A. 3 : Mettre à la valeur `True` les variables d'activation des éléments de type *sequence* ou *message* qui partent de cette tâche.

. C.A. 4 : Rendre cet élément visible à travers la propriété `g:visible`.

. C.A. 5 : Mettre la propriété g :visible de cet élément à la valeur `True` et et mettre à la valeur `True` la variable d'activation spécifique au *StartEvent* contenu dans cet élément.

. C.A. 6 : Mettre à la valeur `True` les variables d'activation des éléments de type *sequence* qui partent de cet élément.

. C.A. 7 : Mettre à la valeur `True` la variable d'activation de l'élément père de type *SubProcess*.

. C.A. 8 : Mettre à la valeur `True` les variables d'activation de l'élement BPMN cible de cet élément.

TABLE 4.3 – Règles de passage du BPM vers PIIM pour remplir la partie `behavior`

BPMN	**Inter**	**Mapp**	**Contrainte** `condition`	**Contenu** `condition`	**Contenu ?**	**Contenu** `action`
User Task	UIFieldIn-Manual	Direct	Cnt.C. 1	C.C. 1	C. 1	C.A. 1
			Cnt.C. 2	C.C. 2	C. 2	
	UIFieldIn-OneChoice	Indica-tive	Cnt.C. 1	C.C. 1	C. 3	C.A. 2
			Cnt.C. 2	C.C. 2	C. 2	
	UIFieldIn-Multi-Choices	Indica-tive	Cnt.C. 1	C.C. 1	C. 4	C.A. 1
			Cnt.C. 2	C.C. 2	C. 2	
	UIField-Navigation	–	Cnt.C. 1	C.C. 1	C. 5	C.A. 1
			Cnt.C. 2	C.C. 2	–	C.A. 3
	UIField-Action	–	Cnt.C. 1	C.C. 1	C. 5	C.A. 1
			Cnt.C. 2	C.C. 2	–	C.A. 3
System Task	UIField-Static	–	–	C.C. 1	C. 6	C.A. 4
	UIField-Out	Indirect	–	C.C. 1	C. 7	C.A. 1
Pool	UIGroup	–	Cnt.C. 5	C.C. 1	C. 8	C.A. 5
	–	–	Cnt.C. 6	C.C. 3		
Sub-Process	–	–	–	C.C. 2	C. 8	C.A. 5
Start Event/ Interme-diate Event	–	–	–	C.C. 1	–	C.A. 6
End Event	–	–	Cnt.C. 3	C.C. 1	–	C.A. 7
			Cnt.C. 4	–		–
Gateway	–	–	–	C.C. 1	–	C.A. 6
Sequence	–	–	Cnt.C. 7	C.C. 4	–	C.A. 8
	–	–	Cnt.C. 8	C.C. 4	–	C.A. 9
Sequence (avec condi-tion)	–	–	Cnt.C. 9	C.C. 5	–	C.A. 9
Message	–	–	Cnt.C. 9	C.C. 1	–	C.A. 9

La Figure 4.12 représente un extrait d'une règle de type "*Called Rule*" du code ATL qui sert à générer la partie `behavior` dans le cas d'un "*UserTask*" auquel est attaché un élément d'interaction de type "*UIFieldInManual*".

```
-- ...
for (task in pool.ContainedElements)
    {
     if (task.ContainedElementType = #UserTask  and task.Related_Static_Element.
         oclIsTypeOf(InteractionModel!UIFieldInManual ))
    {
       for ( mapp in Mapping!Mapping.allInstancesFrom('IN1'))
       {
   if (task.RelatedOntologyElement.OntologyElementName = mapp.Source.
       Ontology_Element_Name  and  mapp.MappingType = #Direct and task.
       RelatedOntologyElement.MappingType = #Direct)

-- Création de la règle UIML (rule)
        rulle <- thisModule.createBehaviorRule('OnlyActivatedrule'+ task.Id.
            toString() , behavior);
        condition <- thisModule.createRuleCondition(rulle);
        op <- thisModule.createConditionOperation('Equal', condition);
        variable <- thisModule.createOperationVariable(task.Id.toString()+ '
            isactivated',op);
        thisModule.createOperationConstant('true',op);
        action <- thisModule.createRuleAction(rulle );
-- Générer la partie "whentrue"
-- la propriété "g:text" sera remplie automatiquement par la valeur déduite du
-- modèle de context à travers la méthode "GetValueFromContext"
        whentrue <- thisModule.createWhenTrueAction(action );
        property <- thisModule.createWhenTruePropertyCall(task.Id.toString(), 'g:
            text',whentrue);
        call <-  thisModule.createPropertyCall(task.Id + 'Context', task.Id + '
            GetValueFromContext', property);
-- le paramètre de la méthode "GetValueFromContext" sera le nom de l'élément de
-- l'ontologie relié à la tâche
        thisModule.createCallParam(task.RelatedOntologyElement.OntologyElementName,
             call);
        thisModule.createWhenTrueProperty(task.Id.toString(), 'g:visible', 'true' ,
            whentrue);
            }}}
-- ...
```

FIGURE 4.12 – Exemple du code ATL (une "*Called rule*" traitant le cas d'un *UserTask*)

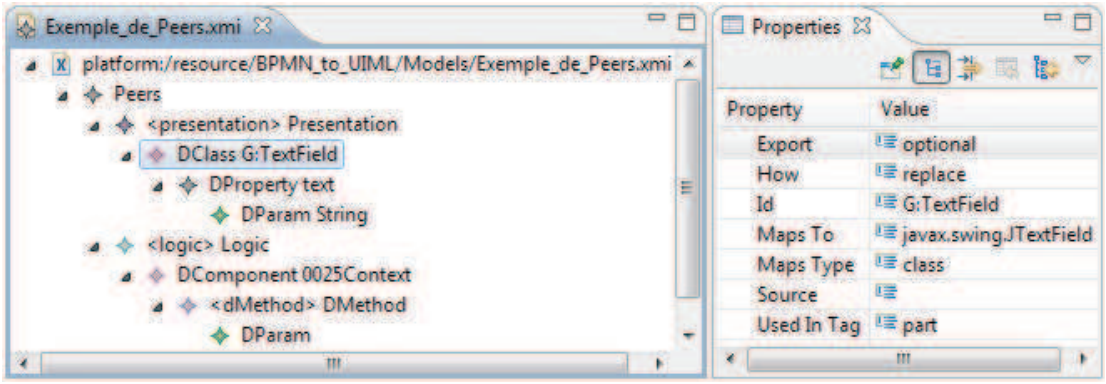
4.3.2 Du PIM vers PSM

Le code UIML générique, généré à l'étape précédente, est à un niveau d'abstraction qui ne lui permet pas d'être transformé en code source, étant donné qu'il n'est pas complet. Afin qu'il puisse être transformé, il doit être spécifique à une plateforme.

Jusqu'à présent, le code UIML générique, obtenu à l'étape précédente, comprend uniquement les parties `structure` et `behavior`. Cette incomplétude ne lui permet pas d'être transformé en code source. De ce fait, il est impératif de compléter ce code UIML et

d'y intégrer les parties manquantes. Ces parties le rendent lié à une plateforme spécifique. Cette étape dans notre approche permettant le passage du PIIM vers le PSIM est nommée une *transition*. En effet, nous ne l'avons pas considéré comme une transformation, étant donné que nous ne générons pas du code à partir du PIIM mais nous intégrons les autres parties manquantes de UIML en fonction de la plateforme cible, à savoir la partie `style` et la partie `peers` (`presentation` et `logic`).

Le développeur peut modifier et enrichir les parties ajoutées, en intégrant davantage d'autres propriétés, en fonction de la plateforme cible. Par exemple, le développeur pourrait éventuellement choisir une couleur particulière pour un composant de l'IHM. La partie `style` au sein de UIML est composée des propriétés qui peuvent être enrichies par les caractéristiques de présentation et de style spécifique à chaque élément de l'IHM. L'addition des parties manquantes au code UIML se fait dans notre approche à travers l'éditeur EMF, qui à travers une opération simple du copier/coller, permet de rajouter les éléments `presentation` et `logic,` préalablement prêts, au modèle UIML obtenu depuis la phase précédente. Quand aux propriétés de la partie `style`, nous utilisons également EMF pour agir sur les éléments du code UIML généré. La Figure 4.13 (a) présente un exemple de la partie `peers` créée avec EMF. Elle peut être intégrée directement dans le code source UIML générique. La partie (b) à droite de la Figure 4.13 représente l'ensemble des propriétés de l'élément `DClass` de type `G:TextField`.



(a) (b)

FIGURE 4.13 – Exemple d'une instance de la partie `peers` créée avec EMF

En ce qui concerne la partie `logic`, pour chaque appel généré dans le cas d'un mapping indirect, dans le PIIM, une balise ≺`logic`≻ sera ajoutée à la partie `peers`, avec les informations sur le code mis en œuvre pour cette méthode. Ce code est implémenté par un développeur afin de rechercher les informations nécessaires en se basant sur un ensemble de paramètres. Pour la partie `presentation`, qui sert à mapper les éléments génériques de UIML vers ceux qui sont spécifiques, il existe plusieurs propositions permettant d'établir ces liaisons. Le Tableau 4.4 présente un exemple de mappings entre le vocabulaire UIML

générique et un ensemble de plateforme spécifiques (Java, HTML 3.2, Palm OS, WML et VoiceXML).

TABLE 4.4 – Correspondances entre le vocabulaire générique de UIML et un ensemble de plateformes spécifiques (Ali et Abrams, 2001)

Élément UIML générique	**Java (swing)**	**Palm OS**	**HTML**	**WML**	**VoiceXML**
G :TopContainer	JFrame	Form	≺HTML≻, ≺Body≻	≺wml≻	≺vxml≻
G :Area	JFrame, Window, JPanel, JScrollPane, JTabbedPane, JTable, JInternalFrame	Window	≺form≻, ≺table≻	≺card≻	≺form≻, ≺table≻
G :Menu	JMenu	Menu	——————— ——	——— —-	≺card≻
G :List	JList	Popup List	≺ul≻, ≺ol≻	≺select≻	——— ——
G :Button	JButton	Command Button	≺input type="button"≻	≺do≻	≺submit≻, ≺clear≻
G :Label	JLabel	Label	≺span≻	≺p≻	≺prompt≻, ≺block≻

4.3.3 De PSM vers le code source

Après avoir préparé le code UIML pour la transformation, il est déterminant de choisir une approche à suivre pour la génération de code ainsi que la technologie à utiliser. Le rendu de UIML (en anglais UIML *rendering*) est le processus de conversion d'un document UIML vers une IHM standard qui peut être présentée à l'utilisateur par le biais d'un système avec lequel l'utilisateur peut interagir (Ali *et al.*, 2002). Le rendu de UIML peut être effectué de deux manières principales :

– Compiler UIML et le transformer vers un autre langage de programmation.

– Interpréter UIML directement et générer immédiatement l'IHM finale.

Si nous utilisons la méthode de compilation, l'IHM qui en résulte est indépendante du moteur du rendu, mais dans le cas d'interprétation il faut prendre compte les évènements d'interaction ainsi que les appels à des fonctionnalités externes à travers les balises `call` de

UIML. Lors de l'interprétation, le document UIML et le moteur du rendu sont nécessaires pour créer et exécuter l'IHM finale. L'avantage de la compilation réside dans la possibilité de modifier et enrichir le code source de l'IHM finale. En suivant cette dernière lignée, nous avons utilisé la méthode de compilation pour transformer le document code UIML en code source en se basant sur l'utilisation de l'outil LiquidApps.

Nous ne pouvons pas apporter plus de détails sur la façon avec laquelle le processus de génération du code est abordé par LiquidApps étant donné que la logique de transformation du code UIML vers du code source n'est pas fournie et représente une boite noire. Nous nous contentons de préparer le fichier UIML (point d'entrée) et ensuite LiquidApps se charge de sa transformation vers un langage cible spécifique.

Conclusion

Dans ce chapitre, nous avons montré l'intérêt de l'instrumentation de notre approche en vue de la modélisation et la génération des IHM à contenus personnalisés. Ce chapitre a présenté une série de décisions méthodologiques et techniques qui nous ont permis de mettre en œuvre notre approche. Jusqu'à présent, nous avons identifié le cadre de métamodélisation de cette proposition, la logique de transformation des modèles, les règles développées pour mettre en œuvre ces transformations et la solution technique permettant de générer le code source de l'IHM finale.

Dans le chapitre suivant, et afin de mettre en pratique notre approche, nous présenterons deux études de cas permettant de générer deux IHM personnalisées. La première application traite un système d'information pour les usagers du transport en commun et la deuxième représente un système d'assistance médicale.

Chapitre 5

Mise en œuvre de l'approche : Études de cas

Introduction

Après avoir présenté notre approche globale et ses différents modèles, nous allons les détailler à travers deux cas d'étude. Le premier traite le domaine de transport en commun et le deuxième présente un système d'assistance médicale. Le choix de ces deux domaines d'application différents permet de mettre en œuvre l'aspect de **généricité** de notre approche et son **indépendance** de tout domaine d'application.

Dans un premier temps, nous allons détailler les différentes étapes permettant de modéliser et générer la première application. Ensuite, dans un second temps, nous appliquons la même démarche pour valider notre approche avec le domaine d'assistance médicale.

5.1 Système d'information pour le transport en commun(1ère étude de cas)

5.1.1 Scénario d'utilisation

Considérons le cas de développement d'un système d'information de planification de voyage pour le transport en commun. Prenons l'exemple du scénario suivant : Alex est un jeune homme âgé de 25 ans et qui souffre d'un handicap l'empêchant de marcher. Il se trouve chez lui à Valenciennes et envisage d'acheter un billet pour aller à Paris. Il se connecte à Internet, par son Mac pour accéder au système d'information des transports en commun. Après sa connexion, le système lui offre trois alternatives. Alex peut choisir celle qui lui convient :

1. La découverte des plans de voyage intéressants ;
2. La recherche des hôtels ; et
3. La recherche instantanée des horaires de départ.

Étant donné qu'il cherche un départ immédiat et qu'il préfère voyager en TGV [1], Alex s'intéresse au dernier choix. En sélectionnant cette option, il reçoit l'espace de travail dédié au départ qui est nommé *Departure form*. Cet espace contient un formulaire, qu'Alex doit remplir avec les informations appropriées de voyage , pour formuler sa demande. À la réception du formulaire, certains champs nommés *ville de départ* (*Departure city*) - *Date et heure de départ* (*Departure date and time*) et *moyen de transport préféré* (*Preferred means of transportation*) ont été remplis automatiquement par le système d'information, d'une manière dynamique en fonction de son contexte d'utilisation.En effet, le champ de la ville de départ est renseigné automatiquement par rapport à la position de l'utilisateur au moment de l'interaction avec la plateforme ; la date de départ correspond à la date d'interaction ; et le choix du TGV, comme moyen de transport, a été déduit du profil d'Alex. Une fois qu'il a validé sa demande, en cliquant sur le bouton de recherche, Alex

1. Un TGV est un **T**rain à **G**rande **V**itesse propulsé par des moteurs électriques.

recevra un tableau contenant les horaires de départ des trains. En prenant en compte, automatiquement, l'âge d'Alex et son handicap, le système lui proposera seulement des itinéraires directs (pour éviter les correspondances entre les gares) avec des prix réduits (selon son âge) ou bien indirects mais sans changement de station.

5.1.2 L'ontologie de transport en commun

La première étape de notre processus de conception consiste à spécifier le mapping entre l'ontologie de domaine et le modèle de contexte. Dans cette étude de cas, nous allons utiliser une ontologie de transport en commun préalablement définie par (Mnasser *et al.*, 2010) et à laquelle nous avons apporté quelques enrichissements (Marcal de Oliveira *et al.*, 2013). Cette ontologie, définie en anglais, présente un ensemble de connaissances sur les transports en commun, portant sur les itinéraires, les points d'arrêt, les villes, les modes de transport utilisés, les éléments géographiques entourant les points d'arrêt (bibliothèques, banques, etc.). La Figure 5.1 montre l'ontologie de transport en commun de haut niveau. Cette ontologie a été formalisée en OWL 1.0 et Protégé. La définition de chaque concept de cette ontologie sera introduite dans le Tableau 5.1.

TABLE 5.1: Glossaire des concepts de l'ontologie de transport en commun

Concepts	**Définitions**
Calendar	Définit une période valable pour un *vehicle journey*.
City	Un centre de population, de commerce et de culture.
Connection Link	La possibilité physique ou spatiale pour un passager de passer d'un véhicule de transport public à un autre pour continuer le voyage.
Connection Point	Un *stop point* dans lequel les passagers changent de véhicule, pour un mode de transport identique ou différent.
Exchange Pole	Un lieu qui facilite les échanges intermodales entre différents modes de transport ; ces échanges se distinguent par la variété des modes de transport réunis en un seul endroit.
Geographic Element	Une lieu, un endroit, une position ou un site.
Geographic Element with services	*Geographic Element* dans lequel les différents magasins offrent aux passagers des services ou bien une structure physique de protection.
Geographic Element without services	*Geographic Element* dans lequel le passager attend un transport en commun.
Infrastructure link	Un lien entre deux points dans un réseau physique.

Infrastructure point	Il peut être un *Railway Junction*, *Wire Junction* ou bien un *Road Junction.*
Journey	Un voyage, depuis une origine jusqu'à une destination, en utilisant un mode de transport spécifique.
Journey pattern	Une liste ordonnée de *Stop point* définissant un seul chemin via le réseau routier ou ferroviaire.
Operator	Les institutions qui offrent le transport en commun
Railway Element	Une sorte de *Infrastructure link* utilisée pour décrire un réseau de chemin de fer.
Railway Junction	Une sorte de *Infrastructure point* utilisée pour décrire un réseau de chemin de fer.
Road Element	Une sorte de *Infrastructure link* utilisée pour décrire un réseau routier.
Road Junction	Une sorte de *Infrastructure point* utilisée pour décrire un réseau routier.
Stop point	Un point dans lequel les passagers peuvent monter ou descendre des véhicules.
Stop point in journey pattern	Un *Stop point* dans lequel le passager ne change pas de véhicules.
Transport line	Un groupe de *Journey pattern* qui est généralement connu du public sous le même nom ou le même numéro.
Transport mode	Une caractérisation de l'opération en fonction des moyens de transport.
Transportation network	Un ensemble de *Transport line* pour assurer le transport en commun.
Vehicle journey	Le mouvement prévu d'un véhicule de transport en commun, sur une semaine, à partir du point de départ jusqu'au point d'arrivée d'un *Journey pattern* sur une infrastructure spécifique.
Vehicle Journey Part	Une partie d'un *Vehicle journey* créée en fonction d'un but fonctionnel spécifique.
Vehicle type	Une classification des véhicules de transport public, selon les exigences du véhicule en termes de mode et de capacité.
Wire Element	Une sorte de *Infrastructure link* utilisée pour décrire un réseau filaire.
Wire Junction	Une sorte de *Infrastructure point* utilisée pour décrire un réseau filaire.

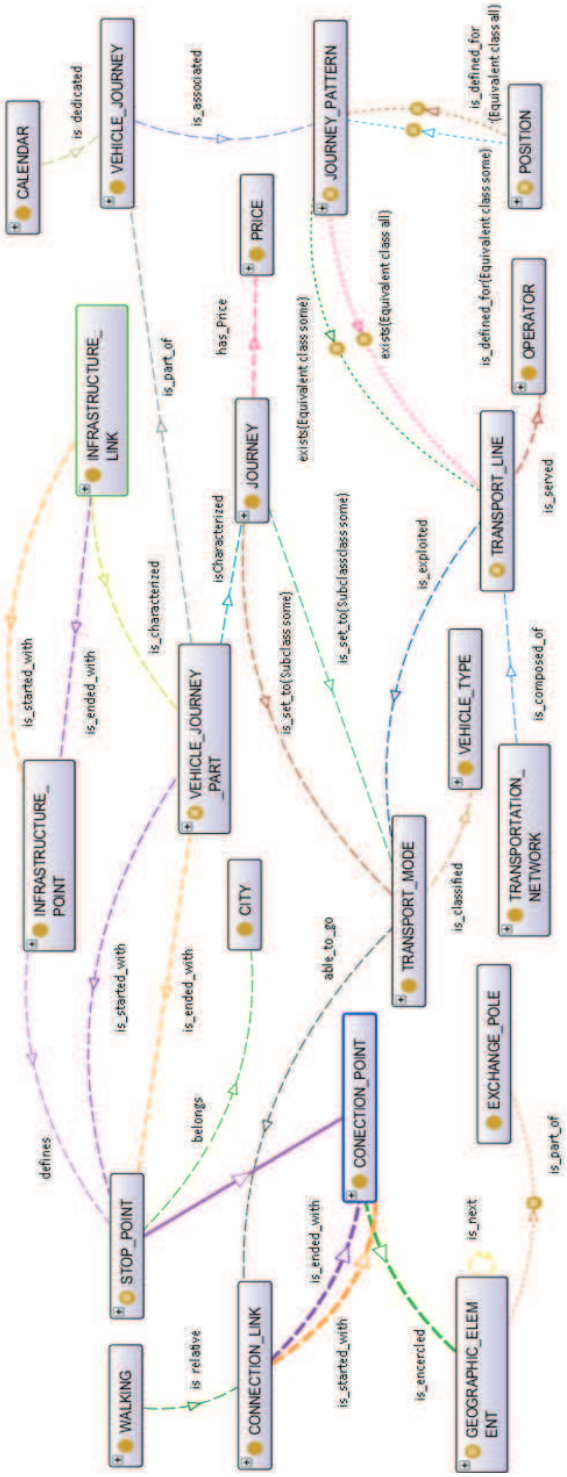


FIGURE 5.1 – Vue globale de l'ontologie de transport

La Figure 5.2 montre un exemple des concepts et des axiomes de l'ontologie ainsi que les relations entre eux. Dans notre cas, nous avons utilisé la notation UML.

Afin de formaliser les classes associatives dans le diagramme UML (*Position* et *Journey*) illustré par la Figure 5.2, (Mnasser *et al.*, 2010) ont utilisé les modèles proposés par (Noy et Rector, 2006) et (Hoekstra, 2009). En utilisant ces modèles, de nouvelles classes et des associations ont été définies pour chaque attribut d'une classe associative (par exemple, pour *Price* and *Duration* et *Rank*).

Pour formaliser l'association de composition (`is_part_of`), (Mnasser *et al.*, 2010) ont utilisé la proposition définie par (Antoniou et van Harmelen, 2004). Le mot *some* est ici équivalent à la restriction existentielle, notée ∃ et le mot *only* est équivalent à la désignation universelle notée ∀. Par contre, le mot *exactly* représente une restriction de cardinalité. Il précise le nombre exact de propriétés qu'un individu peut avoir. Ainsi, nous pouvons citer que :

```
POSITION is_defined_for only JOURNEY_PATTERN

POSITION is_defined_for some JOURNEY_PATTERN

POSITION hasRank only RANK STOP_POINT is_designated only POSITION


JOURNEY is_set_to only TRANSPORT_MODE

JOURNEY is_set_to some TRANSPORT_MODE


JOURNEY has_Price only PRICE

JOURNEY has_Duration only DURATION


VEHICLE_JOURNEY_PART is_characterized_by only JOURNEY

VEHICLE_JOURNEY_PART is_part_of only VEHICLE_JOURNEY


VEHICLE_JOURNEY is_associated exactly 1 JOURNEY_PATTERN

VEHICLE_JOURNEY is_valid_for only CALENDAR

VEHICLE_JOURNEY is_valid_for some CALENDAR


STOP_POINT belongs exactly 1 CITY
```

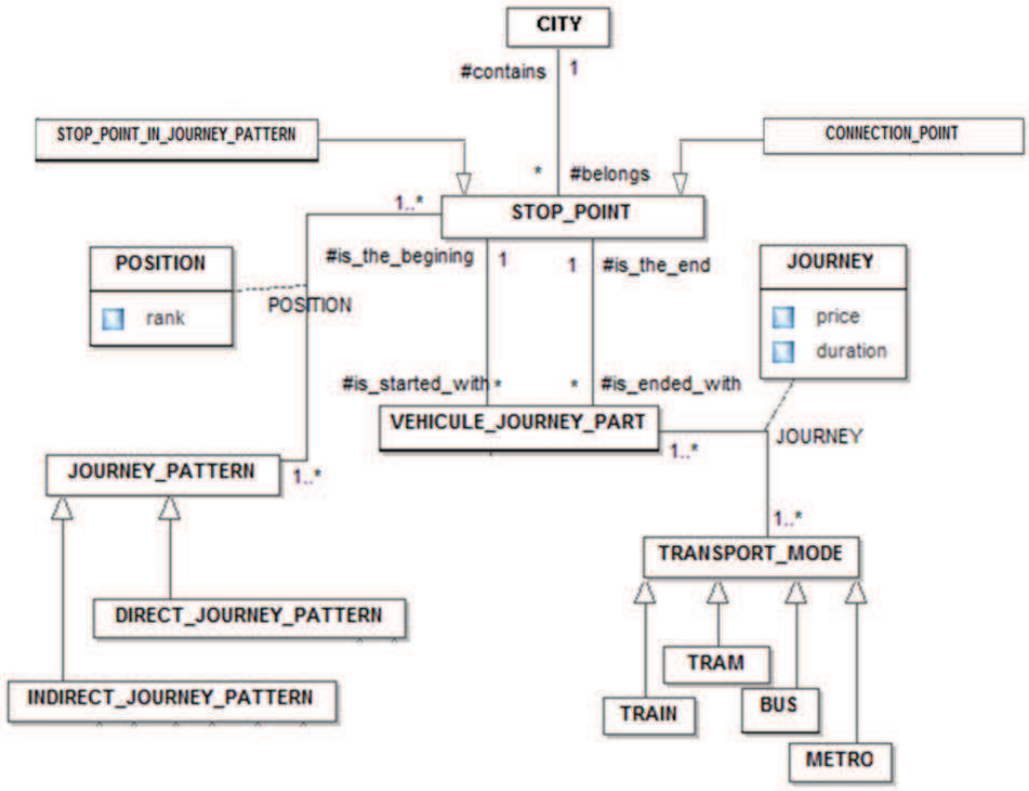


FIGURE 5.2 – Partie de l'ontologie de transport en commun spécifiée en UML

5.1.3 Conception du système d'information pour le transport en commun

Dans cette section, nous présentons la manière avec laquelle le système d'information pour le transport en commun a été conçu afin de prendre en compte la personnalisation du contenu (Bacha *et al.*, 2011b).

5.1.3.1 Mapping entre l'ontologie de transport et le modèle de contexte

L'utilisation de l'ontologie de domaine nous permet de définir le modèle des tâches qui décrit l'IHM au niveau CIM, à travers les concepts de cette ontologie. Ces concepts représentent l'ensemble des connaissances sur le domaine du transport. Toutefois, afin de fournir une IHM personnalisée, nous devons identifier les correspondances existantes entre l'ontologie de domaine et le modèle de contexte.

Comme présenté dans le chapitre 3, cette correspondance est définie sous forme d'un modèle, nommé modèle de mapping dans lequel chaque concept de l'ontologie doit être

mappé, si possible, avec un élément du modèle de contexte, pouvant l'influencer. La Figure 5.3 représente le modèle de contexte spécifique au domaine de transport en commun. La manière permettant d'obtenir un tel modèle, à partir de notre métamodèle de contexte, a été présentée dans le chapitre3, §3.2.1.

En analysant l'ontologie de transport en commun (cf. Figure 5.1) et le modèle de contexte de transport (cf. Figure 5.3), nous pouvons identifier les trois types de mappings :

- **Le mapping direct :** dans ce premier type de correspondances nous nous référons à des concepts qui représentent exactement les mêmes informations présentes dans le modèle de contexte, bien que parfois avec un nom différent. Dans cette étude de cas, le concept de l'ontologie de transport *City* représente la même information fournie dans la classe *contact information* (i.e. l'attribut *city*), ce qui signifie que l'information de contexte a une influence directe sur le contenu de la même information de l'ontologie. En plus l'attribut *city* est de type "string".

- **Le mapping indicatif :** ce deuxième cas se rapporte aux attributs du modèle de contexte, qui peuvent influencer certains concepts de domaine définissant l'existence ou l'absence de certaines informations. Par exemple, l'attribut TGV du modèle de contexte indique la préférence de l'utilisateur pour certains types de trains qui représentent un concept de l'ontologie (la classe *Train*). De la même manière, le concept de l'ontologie *Bus* pourrait être mappé à l'attribut *bus* du contexte, en utilisant un mapping indicatif. Notons bien que les attributs du modèle de contexte TGV et *bus* sont de type "Boolean".

- **Le mapping indirect :** ce troisième cas se réfère à certains nombre de concepts du modèle de contexte qui peuvent avoir une influence indirecte sur les concepts de l'ontologie. Ceci signifie qu'ils peuvent avoir indirectement un impact sur le contenu du concept du domaine. Dans notre étude de cas, l'information sur l'âge de l'utilisateur dans le modèle de contexte (*age*) peut influencer le prix du billet (représenté par le concept *Price*). Un autre exemple du mapping indirect est l'information de la capacité de l'utilisateur de marcher ($ToWalk$) qui est indirectement associé au concept *Walking* de l'ontologie. Ainsi, la possibilité de pouvoir se déplacer entre les arrêts à pieds est fortement liée à l'aptitude du voyageur à marcher. Le dernier mapping indirect dans notre exemple est réalisé entre les concepts *Shelter* et *Platform* de l'ontologie du domaine et l'attribut *Raining* appartenant au modèle de contexte. En effet, un parcours protégé de la pluie doit forcément contenir des abris.

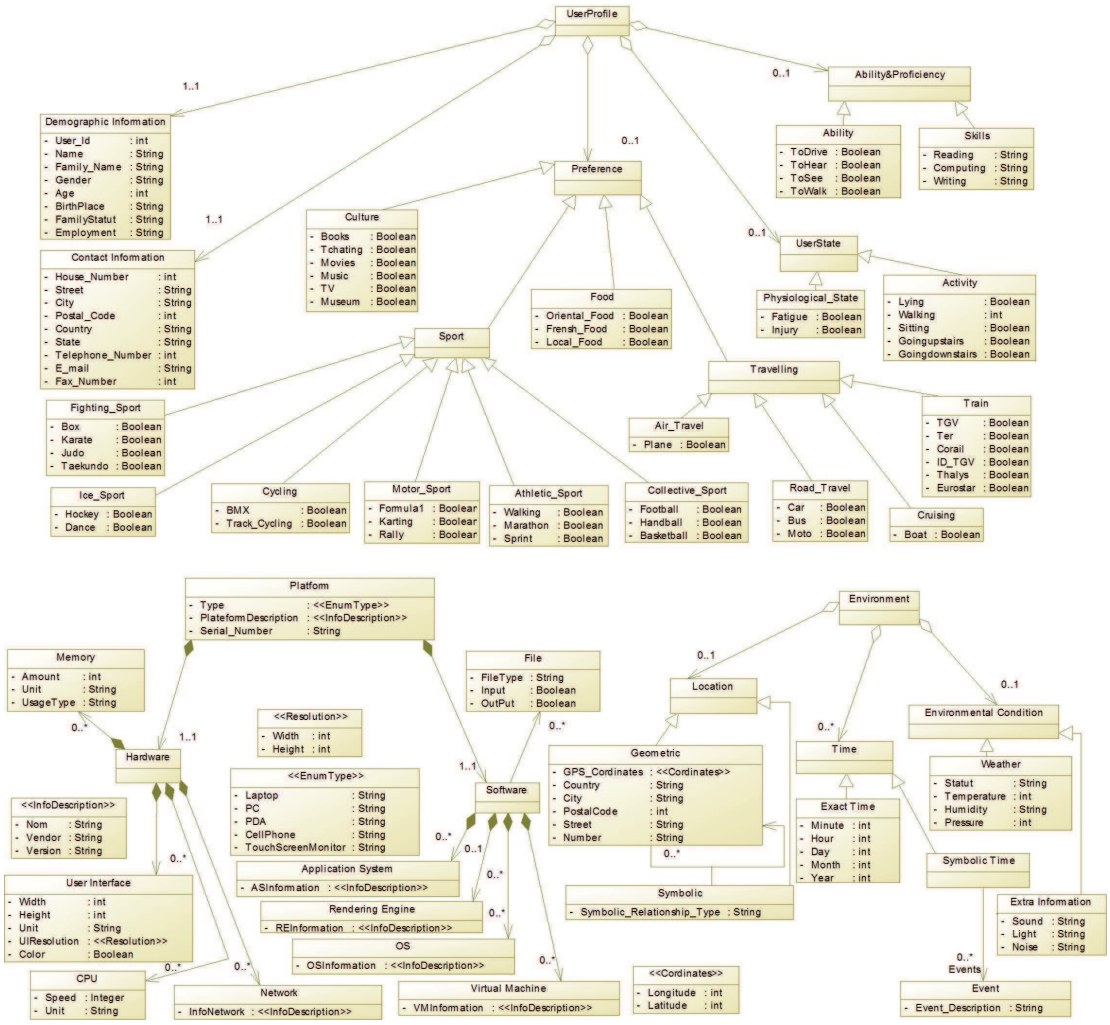


FIGURE 5.3 – Modèle de contexte spécifique au domaine de transport en commun

La Figure 5.4 montre quelques exemples de mappings entre l'ontologie de transport en commun et les éléments du modèle de contexte.

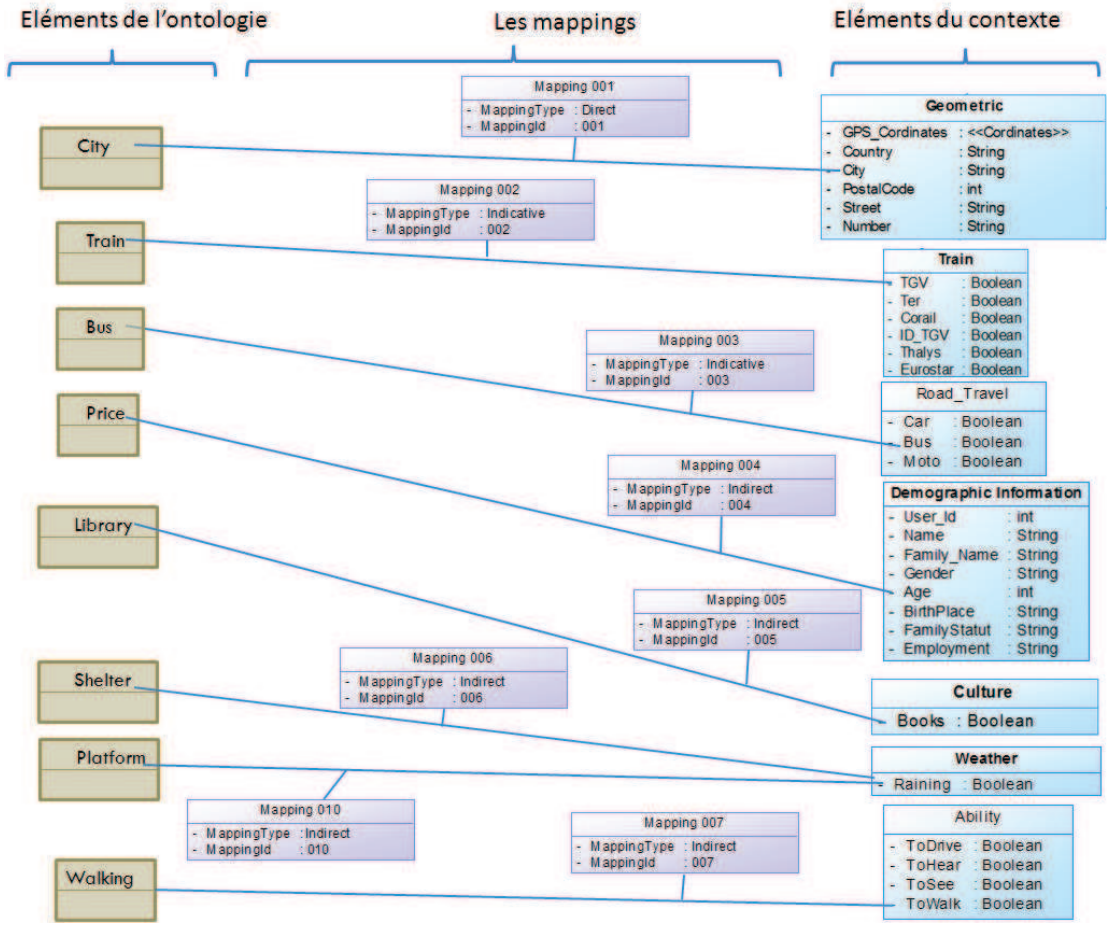


FIGURE 5.4 – Modèle de mapping entre l'ontologie de transport et le modèle de contexte

Les mappings définis pour un domaine spécifique peuvent être utilisés dans le développement de plusieurs applications dans le même domaine. Une fois les mappings élaborés, nous pouvons les utiliser dans la définition du modèle des tâches pour décrire le système d'information de transport en commun. Ci-dessous, nous présentons les étapes suivies pour élaborer ce modèle et la façon selon laquelle nous utilisons ces mappings afin d'assurer la personnalisation du contenu dans ce système.

5.1.3.2 Étape 1 : Élaboration du modèle des tâches

Nous allons décrire dans ce paragraphe la mise en œuvre du modèle des tâches traduisant le scénario précédent, en utilisant la version étendue de BPMN proposée. La Figure 5.5(a) illustre la première partie du scénario qui définit l'ensemble des choix fournis à l'utilisateur au niveau du formulaire de départ. Dans la Figure 5.5(b), nous étendons le sous processus (Subprocess) décrivant le formulaire de départ, afin d'illustrer le reste du scénario. En suivant notre approche de modélisation, la conception de l'IHM se fait en deux grandes étapes : dans la première, le concepteur élabore un modèle des tâches non annoté puis, dans la seconde, il annote ce modèle avec les informations requises afin d'y apporter une personnalisation du contenu.

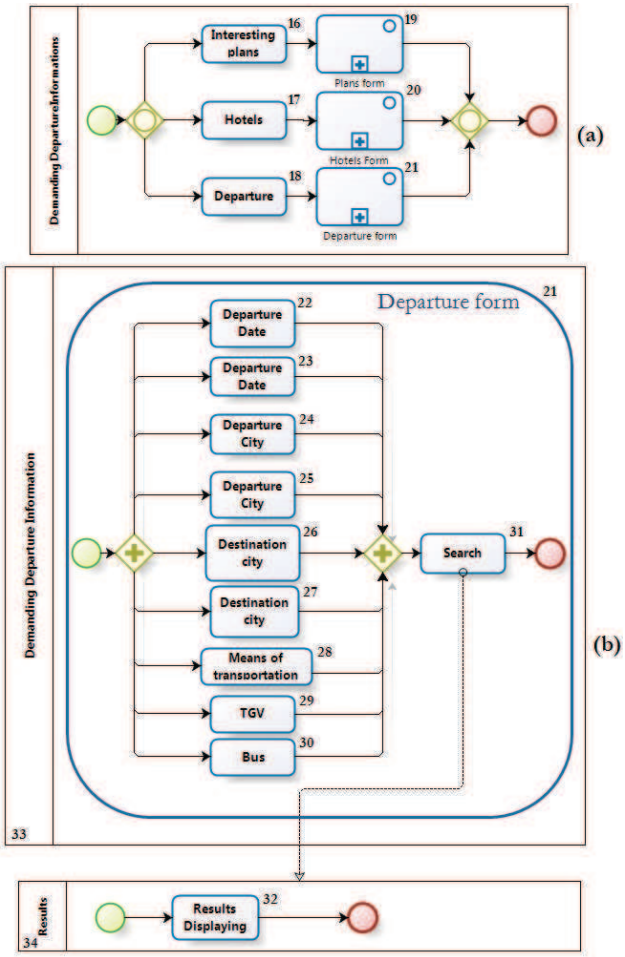


FIGURE 5.5 – Modèle des tâches modélisant le système d'information pour le transport en commun

Afin de créer un modèle des tâches non annoté, il faut que :

- Chaque élément de type *"Pool"* soit utilisé comme une entité organisationnelle pour faire référence à un espace de travail de l'IHM finale. Étant donné que le modèle précédent contient deux espaces de travail - "Departure Information" et "Results", il nous faudra donc deux *"Pools"*.
- Chaque espace de travail est composé de plusieurs éléments (Tasks – Events – Gateways etc.). Le concepteur doit commencer par modéliser les activités ("Task" et *"Subprocess"*) appartenant à chaque espace. Pour toute tâche (*task*), le concepteur précise s'il s'agit de *UserTask* ou un *SystemTask*. Et afin de synchroniser tous les éléments précédents, il peut utiliser les *"events"*, les *"gateways"* ainsi que les *"arcs"*. Notons que les tâches de (22) à (30) peuvent avoir lieu simultanément. Pour mettre en œuvre cet aspect, nous avons utilisé l'élément *"Parallel Gateway"* lors de la dissociation et lors de l'association des *"sequence flow"*. Une fois ces tâches terminées, l'utilisateur peut lancer le processus de recherche des résultats (tâche numéro (31)). L'élément de type *"message flow"* sortant de la tâche numéro (31) et rejoignant le *"Pool"* numéro (34) correspond à un changement d'espace de travail lors du lancement de la recherche. Au moment de la réception de ce message, le processus inclus dans le *"Pool"* numéro (34) peut débuter. Dans ca cas d'application, il est composé de la tâche (32) permettant l'affichage des résultats de recherche.

5.1.3.3 Étape 2 : Annotation du modèle des tâches

Une fois que nous avons défini un modèle des tâches non annoté, nous devons l'annoter avec des informations pouvant agir sur la personnalisation du contenu, lors de l'exécution. L'idée est d'introduire dans ce modèle, les préférences de l'utilisateur pour la ville de départ, pour la date du voyage et pour le mode de transport que le système est censé connaitre. Grâce à ces informations, le système peut fournir un formulaire pré-rempli à l'utilisateur. Bien que l'utilisateur peut modifier ces informations, le système doit présenter le formulaire avec tous les contenus recueillis. Les résultats de la requête de la recherche doivent tenir compte de la connaissance du domaine et du contexte. Dans cet exemple, l'utilisateur est incapable de marcher. Ainsi, le système doit proposer des itinéraires directs avec des prix réduits, selon son âge. Dans le cas des itinéraires indirects, le système ne doit proposer que des parcours protégés contenant des abris, car il peut pleuvoir. À cette fin, après avoir conçu le modèle de tâche non annoté, le concepteur peut annoter, si nécessaire, chaque *"Pool"*, *"task"*, *"SubProcess"* ou bien *"Group"*, afin de prendre en compte la personnalisation du contenu dès la phase de modélisation de l'IHM.

- Tout d'abord, et étant donné que chaque espace de travail contient un certain nombre d'éléments qui le construisent, chaque composant de l'interface sera associé à la tâche qui le manipule. À cette fin, le concepteur doit associer à chaque élément BPMN, l'élément d'interaction approprié. En suivant les règles d'association présentées précédemment (cf. chapitre 3, §3.3.2), le Tableau 5.2 résume l'annotation de notre exemple de modèle BPMN avec des éléments du modèle d'interaction.

TABLE 5.2 – Annotation du modèle des tâches par les éléments d'interaction (cas du système d'information pour le transport en commun)

Identifiant de l'élément BPMN	**Type de l'élément BPMN**	**Élément d'interaction associé**
33, 34	Pool	UIGroup
22, 24, 26, 28	SystemTask	UIFieldStatic
23, 25, 27	UserTask	UIFieldInManual
29, 30	UserTask	UIFieldOneChoice
31	UserTask	UIFieldAction
32	SystemTask	UIFieldOut
19, 20,21	SubProcess	UIUnit
16, 17, 18	UserTask	UIFieldNavigation

Après avoir associé les éléments d'interaction aux différents composants du modèle des tâches, et en vue d'appliquer la totalité des règles d'annotation du modèle BPMN, présentées dans le chapitre 3, §3.3, on procède comme suit :

- Sur la base de la règle 1 et des mappings existants, nous associons à la tâche (25) le concept de l'ontologie *City* via l'attribut *OntologyElementName* et nous choisissons l'option directe (pour l'attribut *mapping type*). Cela signifie que la valeur du champ de saisie "Departure City" correspond à la valeur de l'élément de contexte mappé directement à cette tâche, dans ce cas à travers l'attribut *city*.
- En fonction de la règle 2, nous associons à la tâche (29) le concept *Train* et à la tâche (30) le concept *Bus* appartenant à l'ontologie. Ensuite, nous précisons au niveau de l'attribut *MappingType* qu'il s'agit d'un mapping indicatif, afin de montrer que la sélection des alternatives TGV ou Bus dépend des préférences de l'utilisateur (s'il préfère ou non ce moyen de transport).
- D'après la règle 3, nous associons à la tâche numéro (32), le concept de l'ontologie recherché, dans ce cas *Journey Pattern* (via l'attribut *OntologyElementName*) et ensuite nous mentionnons les paramètres qui doivent être pris en compte pendant sa recherche. Dans ce cas, ce sont les classes *Library* et *Walking*. Étant donné que le concepteur connait déjà les mappings cela signifie que lors de la remise du résultat de la recherche, si l'utilisateur est incapable de marcher, le système ne doit lui fournir que des parcours directs et doit également prendre en compte le fait qu'il est intéressé par les bibliothèques et par la lecture.

La Figure 5.6 résume l'ensemble des annotations apportées aux éléments du modèle des tâches afin de prendre en compte la personnalisation du contenu lors de la phase de modélisation.

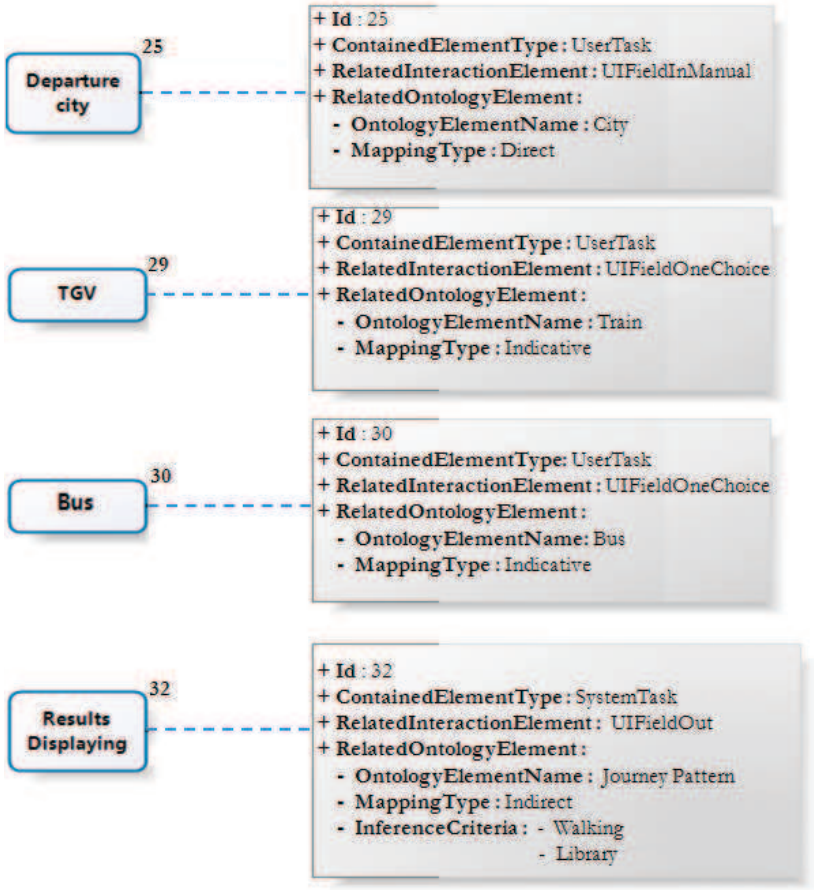


FIGURE 5.6 – Annotation des éléments du modèle des tâches (cas du système d'information pour le transport en commun)

Une fois le modèle des tâches annoté défini, le concepteur peut lancer le processus de transformation de modèles. Dans la prochaine section, nous allons décrire le code UIML généré pour les niveaux PIM et PSM.

5.1.4 Génération du code UIML

Comme mentionné précédemment, nous avons utilisé ATL pour coder les transformations de nos modèles. Ces transformations utilisent en entrée le métamodèle de mapping, le modèle de contexte, l'ontologie de domaine, le modèle d'interaction ainsi que le modèle des tâches. En se basant sur ces modèles, nous générons au niveau PIM trois parties du code UIML, à savoir `structure`, `behavior` et `style`. Dans le code UIML généré, la personnalisation du contenu au niveau de l'IHM se fait à la base de deux méthodes : le remplissage automatique des formulaires et l'enrichissement des requêtes.

Le remplissage automatique des formulaires est effectué pour tous les champs dotés d'un mapping direct ou indicatif dans le modèle des tâches (cf. Figure 5.6). Dans le cas de notre exemple, pour le mapping direct, la propriété `g:text` qui contient le contenu de l'élément numéro 25 est remplie automatiquement avec la valeur de l'attribut *city* et elle est déduite de l'instance du modèle de contexte en appelant la méthode `0025GetValueFromContext`. Le code UIML généré suivant, montre la manière avec laquelle le champ ville de départ (*Departure city*) va être rempli par la valeur de l'attribut *city.*

```
<action>
  <whenTrue>
    <property name= "g:text" partName= "0025">
     <call componentId= "0025Context" methodId= "0025GetValueFromContext">
     <param name= "City"/>
     </call>
    </property>
  </whenTrue>
</action>
```

De la même manière, nous traitons les concepts de l'ontologie dotés des mapping indicatifs. Afin de décider de la sélection ou non de l'élément concerné, en fonction des préférences de l'utilisateur, la valeur de l'élément de contexte (`true` / `false`) est vérifiée dans la partie `condition` de UIML (via la propriété `g:selected`). La partie suivante du code est générée pour traiter le cas du TGV. Elle aura la pré-sélection de la bonne alternative en se basant sur les préférences de l'utilisateur :

```
 ...
<variable name= "variable0080"
  <property>
    <call componentId= "0029Context" methodId= "0029GetValueFromContext">
     <param name= "TGV"/>
    </call>
  </property>
</variable>
 ...
<whenTrue>
  <property name= "g:selected" partName= "0029">
   <constant value= "true"/>
 </property>
</whenTrue>
 ...
```

Pour les mappings indirects, la transformation génère au niveau PIM la partie `call` de UIML qui représente une abstraction de tout type d'invocation de méthodes externes. En effet, il s'agit d'une méthode qui doit être codée par le concepteur de logiciels pour rechercher les informations nécessaires en se basant sur les données passées en paramètres. Cette méthode est équivalente à une requête de recherche dans laquelle la première balise présente l'élément recherché (dans notre cas *Journey pattern*) et la suite des balises présente les paramètres à prendre en compte lors de cette recherche. Comme nous sommes intéressés par fournir un contenu personnalisé pour l'utilisateur, cette requête a été enrichie par l'inclusion automatique d'autres éléments de l'ontologie. Cet enrichissement s'effectue en respectant les règles présentées dans le chapitre 3. Le Tableau 5.3 résume la manière

avec laquelle l'ensemble des relations et des axiomes de l'ontologie de transport en commun a été exploité pour appliquer ces règles.

TABLE 5.3 – Enrichissement de la requête en fonction des axiomes de l'ontologie de transport en commun

Caractéristique de la relation	**Comment exploiter l'axiome ?**
FunctionalProperty	Pour rechercher l'élément *Journey Pattern*, le concepteur définit la classe *Walking* comme un *InferenceCriteria*. Parmi les éléments faisant partie du chemin reliant ces deux classes, la transformation détecte la classe *Journey*. Cette classe possède une propriété fonctionnelle (*has_Price*) qui la relie avec la classe *Price*, dotée d'un mapping indirect. Ainsi, la classe *Price*, sera automatiquement ajoutée à *InferenceCriteria* pendant la recherche du *Journey Pattern* (cf. Figure 5.7).
SymmetricProperty	Pour rechercher l'élément *Journey Pattern*, le concepteur définit les classes *Walking* et *Library*. En vérifiant les propriétés qui relient ces classes, la seule propriété symétrique trouvée est *is_next* qui relie la classe *Geographic_Element* à elle même. Par conséquent, dans ce cas, aucune nouvelle classe ne sera ajoutée à *InferenceCriteria* (cf. Figure 5.7).
TransitiveProperty	Pour rechercher l'élément *Journey Pattern*, le concepteur définit la classe *Library* comme *InferenceCriteria*. Comme la classe *Shelter* est dotée d'un mapping indirect et comme la classe *Geographic_Element* est une super-classe pour *Library* et *Shelter*, ayant une propriété transitive (*is_next*), alors *Shelter* sera automatiquement ajoutée à *InferenceCriteria* pendant la recherche du *Journey Pattern* (cf. Figure 5.7). De même, la classe *Platform* est dotée d'un mapping indirect et elle constitue une sous-classe de *Geographic_Element*. Par conséquent, *Platform* est également ajouté à *InferenceCriteria* pendant la recherche du *Journey Pattern* (cf. Figure 5.7).

Le code UIML suivant, généré au niveau du PIIM, dans le cas des mappings indirects comprend une méthode appelée `Get-Element`. Cette dernière est appelée via l'instruction `call`. Ses paramètres sont l'élément recherché (*Journey Pattern*) et les paramètres que le système devrait prendre en considération lors du processus de recherche à savoir les classes

Walking, *Shelter*, *Platform*, *Library* et *Price*.

```
<action>
  <whenTrue>
   <property name="g:text" partName="0032">
    <call componentId="0032Context" methodId="0032Get-Element">
     <param name="Journey Pattern"/>
     <param name="Library"/>
     <param name="Walking"/>
     <param name="Platform"/>
     <param name="Price"/>
     <param name="Shelter"/>
    </call>
   </property>
  </whenTrue>
</action>
```

Après avoir généré le code UIML du PIIM, le concepteur doit intégrer les parties manquantes de UIML afin d'obtenir un code complet pouvant être interprété ou transformé vers une plateforme spécifique. Le code UIML présenté ci-après montre une partie du résultat de la transformation de PIIM de PSIM en supposant que la plateforme cible utilise le langage Java. En effet, la classe `G:TextField` générée au niveau du PIIM, sera mappée avec la classe `JTextField` de la bibliothèque Swing. La méthode générée nommée `0025GetValueFromContext` est associée à la méthode `Lamih.Context.GetValueFromContext` qui est spécifique à la plateforme cible et qui permet d'extraire des informations de contexte. La méthode `0032GetElement` est mappée à la méthode `Lamih.Context.GetValueFromContext` qui permet de rechercher un élément (`param1_32`) en considérant cinq critères (`param2_32`, `param3_32`, `param4_32`, `param5_32`, `param6_32`).

```
<UIML:Peers id= "MainPeers">
 <UIML:Presentation id= "MainPresentationPart">
  <d-class id= "G:TextField" used-in-tag= "part" maps-type= "class" maps-to= "javax
      .swing.JTextField">
   <d-property id= "text" maps-type= "setMethod" maps-to= "setText">
    <d-param type= "java.lang.String"/>
   </d-property>
  <d-class>
 </UIML:Presentation>
 <UIML:logic id= "MainLogicPart">
  <dComponent id= "0025Context" mapsTo= "Lamih.Context">
   <dMethod id= "0025GetValueFromContext" mapsTo= "Lamih.Context.
       GetValueFromContext">
    <dParam id= "param0025GetValueFromContext" type= "String"/>
   </dMethod>
  </dComponent>
  <dComponent id= "0032Context" mapsTo= "Lamih.Context">
   <dMethod id= "0032Get-Element" mapsTo= "Lamih.Context.Get-Element">
    <dParam id= "param1_32" type= "String"/>
    <dParam id= "param2_32" type= "String"/>
    <dParam id= "param3_32" type= "String"/>
    <dParam id= "param4_32" type= "String"/>
    <dParam id= "param5_32" type= "String"/>
    <dParam id= "param6_32" type= "String"/>
   </dMethod>
  </dComponent>
 </UIML:logic>
</UIML:Peers>
```

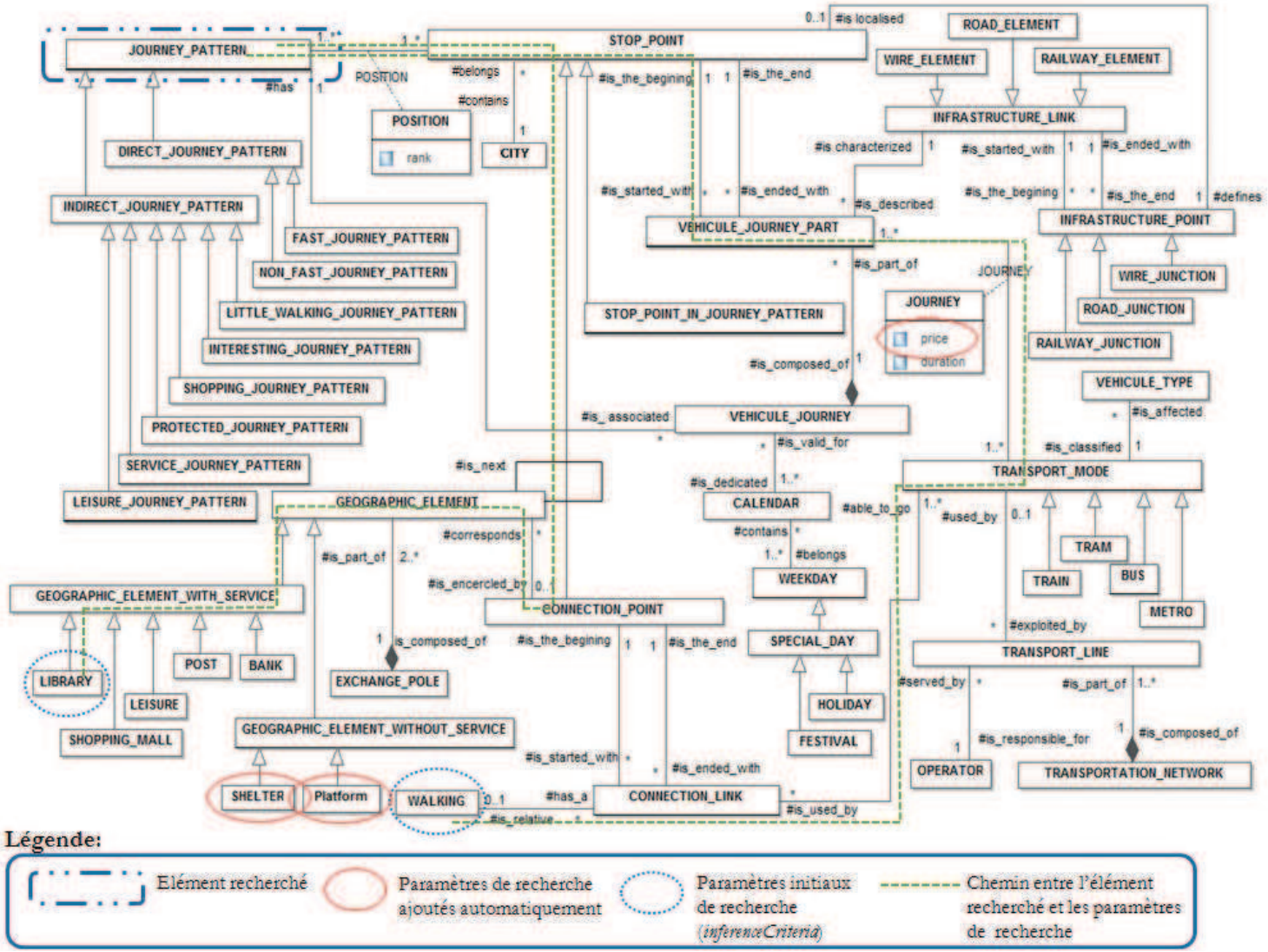


FIGURE 5.7 – Enrichissement des requêtes dans le cas de l'ontologie de transport en commun

La Figure 5.8 illustre quelques exemples de personnalisation du contenu et de contenant d'une éventuelle IHM correspondant au code UIML obtenu. Étant donné que l'utilisateur se sert de son Mac lors de l'interaction avec le système et qu'il est à son domicile à Valenciennes (cf. Figure 5.8 / (a) et (b)), la ville de départ est remplie automatiquement en se basant sur l'emplacement actuel de l'utilisateur. La date de départ est également automatiquement remplie par la date actuelle au moment de l'interaction. Puisque le système connait que l'utilisateur préfère voyager en TGV, l'option TGV sera pré-sélectionnée. Un autre exemple de la personnalisation du contenu est présenté dans la Figure 5.8 (b). Dans cet exemple, l'utilisateur est incapable de marcher ; et par conséquent, le système lui proposera uniquement des itinéraires directs avec des prix réduits (selon son âge).

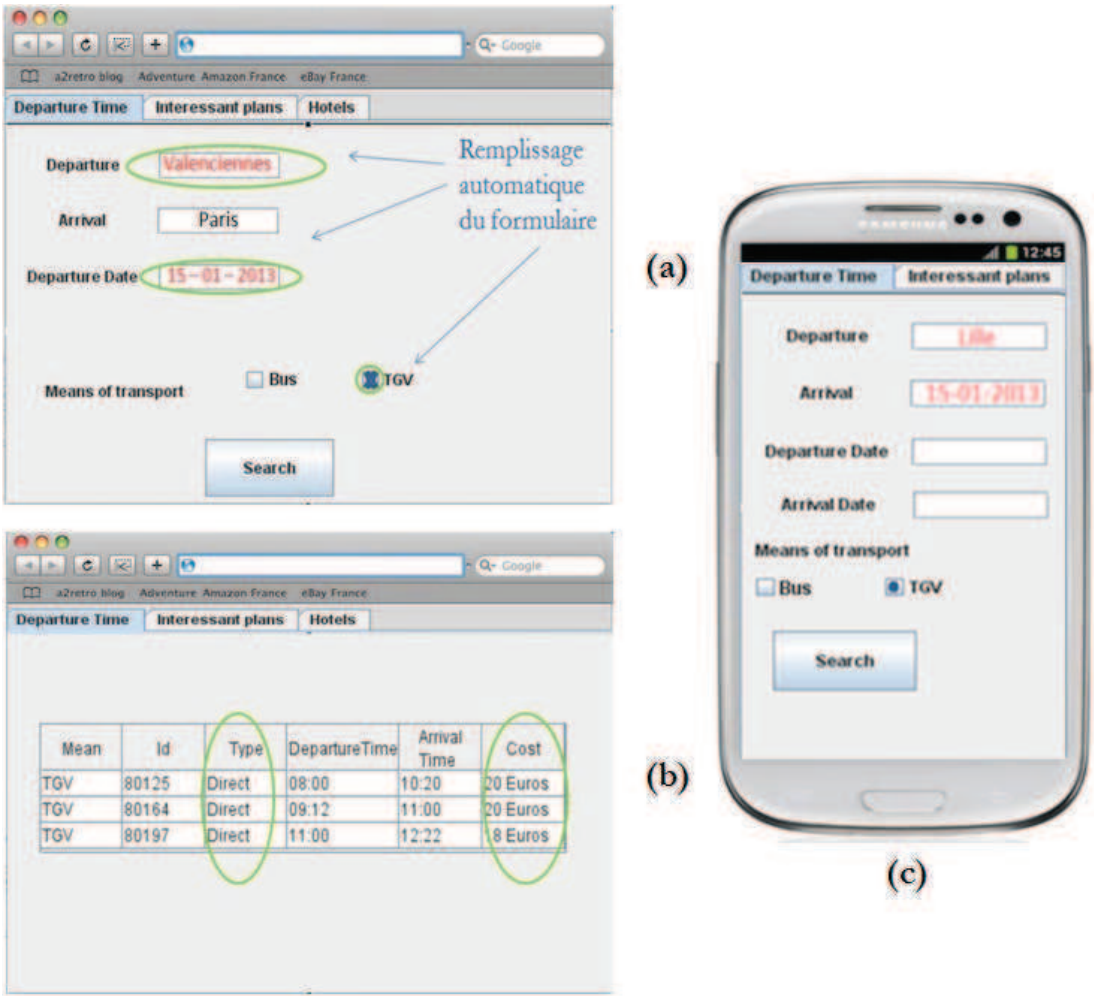


FIGURE 5.8 – Exemples d'IHM générées pour le système d'information pour le transport en commun

La Figure 5.8 (c) montre la même IHM accessible depuis un autre appareil (un smartphone) à partir d'une autre ville (Lille). Afin de mettre en œuvre l'adaptation du contenant, les tailles des éléments de l'interface tels que les polices et les widgets peuvent être adaptées en fonction de la plateforme cible (dans ce cas, un Mac et un smartphone). Par exemple, étant donné que la taille de l'écran du smartphone est inférieure à celle d'un Mac, l'arrangement et les tailles des éléments composant l'interface ont été adaptés à la nouvelle plateforme.

5.2 Système d'assistance médicale (2ème étude de cas)

5.2.1 Scénario d'utilisation

Pour une meilleure explication de notre approche, nous détaillerons dans ce qui suit, via cette deuxième étude de cas, les différentes étapes permettant de générer un système d'assistance médicale. Cette application a pour but de fournir aux utilisateurs un ensemble d'informations médicales en fonction de leur état de santé. A partir des informations sur le contexte de l'utilisateur, explicitement ou bien implicitement fournies (symptômes, environnement, genre, âge, etc.), le système fournira à l'utilisateur une suggestion de conseils à sa maladie.

L'IHM de ce système est composée de deux espaces de travail. Dans le premier espace, le système demande d'abord à l'utilisateur de remplir un formulaire en lui communiquant certaines informations telles que la région du corps atteinte par la maladie (par exemple la tête, la gorge, etc.), l'âge et les symptômes prédéfinis (par exemple fièvre, fatigue, douleur oculaire, toux, vertiges, tremblements). Notons qu'au moment de la réception du formulaire, certains champs peuvent être remplis automatiquement par le système. Une fois que l'utilisateur valide sa demande, le second espace s'ouvrira sur les résultats de sa recherche. Ces résultats sont sous la forme d'un tableau qui liste l'ensemble des maladies dont l'utilisateur pourrait être atteint. Pendant la recherche des résultats, le système considère non seulement les informations fournies explicitement par l'utilisateur mais aussi d'autres paramètres déduits du contexte au moment de l'interaction.

5.2.2 L'ontologie des maladies

La première étape de notre processus de conception proposée est la spécification du mapping entre l'ontologie de domaine et le modèle de contexte. Dans cette deuxième étude de cas, nous allons utiliser une ontologie générique définie par (Hadzic et Chang, 2005) pour décrire les maladies humaines . Cette ontologie que nous avons enrichie est composée principalement de 6 concepts :

– La maladie (*Disease*) : altération de la santé, des fonctions des êtres vivants (animaux et végétaux), en particulier quand la cause est connue (par opposition à syndrome) [2].

2. http ://www.larousse.fr

– Le patient (*Patient*) : la personne soumise à un examen médical, qui suit un traitement ou subit une intervention chirurgicale.
– Les types (*Types*) : les différentes catégories des maladies.
– Les causes (*Causes*) : les origines ou les causes responsables des maladies.
– Les symptômes (*Symptoms*) : les manifestations d'une maladie, la plainte du patient.
– Les traitements : les manières de remédiation des maladies.

La Figure 5.9 montre une vue globale de l'ontologie des maladies représentant l'ensemble des classes de l'ontologie ainsi que les relations reliant celles-ci. Cette ontologie a été formalisée en OWL1.0 et Protégé 4.1.

Afin de mieux expliquer cette ontologie, nous essayons dans ce qui suit de décrire l'ensemble des contraintes (axiomes) appliquées sur les concepts et les relations :

```
DISEASE affects some PATIENT

DISEASE is_caused_by some CAUSES

DISEASE is_caused_by some CAUSES

DISEASE has only TYPES

DISEASE is_cured_by some TRAITMENTS

PATIENT is_affected_by some DISEASE

PATIENT has_age only AGE

PATIENT has_gender only GENDER

PATIENT shows some SYMPTOMS

CAUSES is_responsible_for some DISEASE

CAUSES is_derived_from some CAUSES

SYMPTOMS characterizes some PATIENT

TYPES is_of some DISEASE

TRAITMENTS cures some DISEASE
```

5.2.3 Conception du système d'assistance médicale

Dans cette section, nous présentons la manière avec laquelle le système d'assistance médicale a été conçu afin de prendre en compte la personnalisation du contenu (Bacha *et al.*, 2011c, 2013).

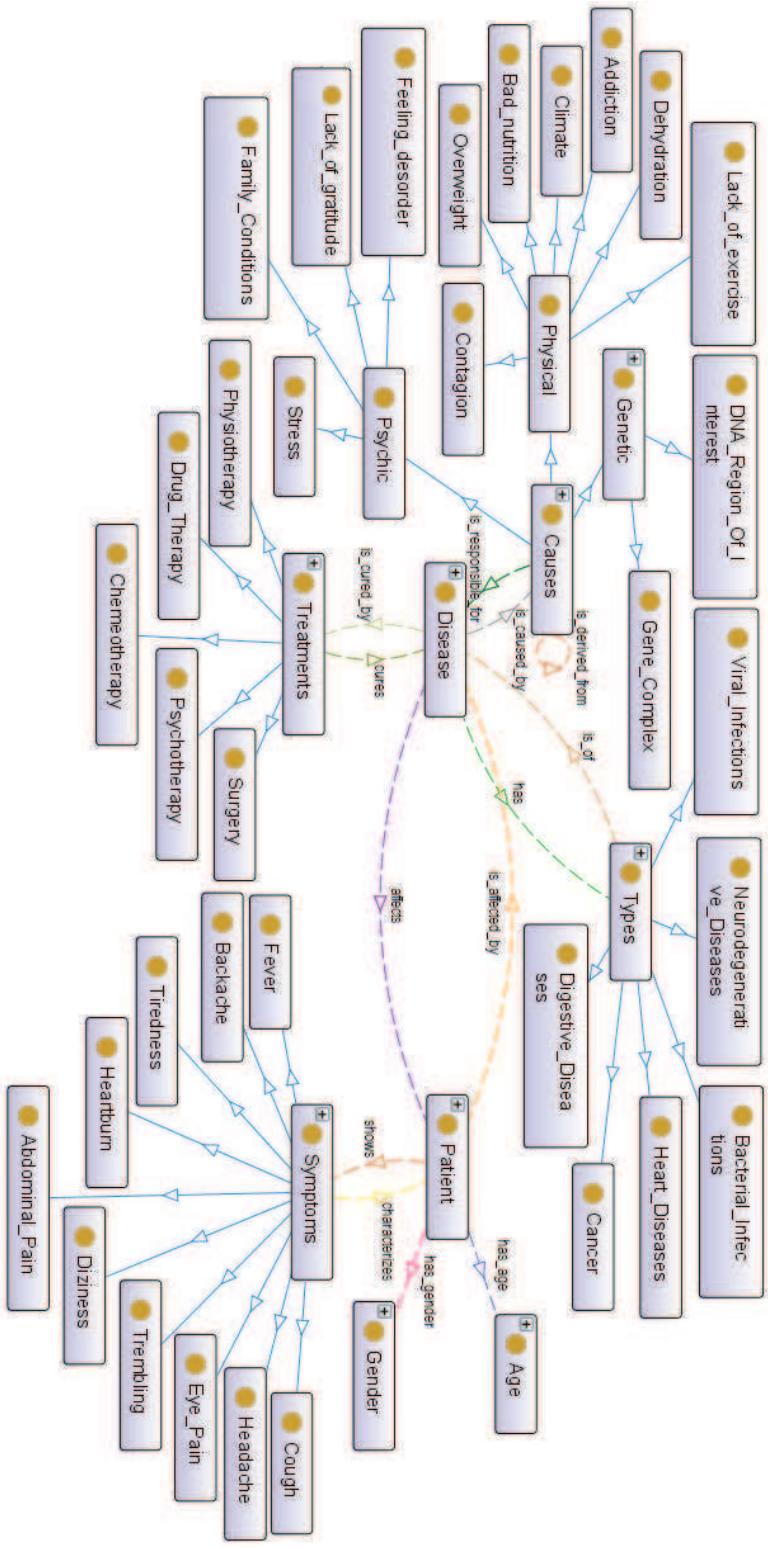


FIGURE 5.9 – Vue globale de l'ontologie des maladies

5.2.3.1 Mapping entre l'ontologie des maladies et le modèle de contexte

Avant de réaliser les mappings entre l'ontologie de domaine et le modèle de contexte, nous commençons par instancier le modèle de contexte générique comme indiqué dans le chapitre 3, §3.2.1, afin d'obtenir un modèle de contexte spécifique. La Figure 5.10 présente le profil d'utilisateur spécifique au domaine de la médecine.

En analysant l'ontologie des maladies (cf. Figure 5.9) et le modèle de contexte de médecine (cf. Figure 5.10), nous pouvons identifier trois types de mappings :

- **Le mapping direct :** dans ce premier type de correspondance, nous nous référons à des concepts qui représentent exactement les mêmes informations présentes dans le modèle de contexte, bien que parfois avec des noms différents. Dans cette étude de cas, le concept de l'ontologie de médecine *Age*, représente la même information fournie dans la classe *Demographic information* (i.e. l'attribut *Age*), ce qui signifie que l'information de contexte a une influence directe sur le contenu de la même information de l'ontologie. En plus, l'attribut *Age* est de type "int".
- **Le mapping indicatif :** ce deuxième type se rapporte aux attributs du modèle de contexte, qui peuvent influencer certains concepts de domaine en définissant l'existence ou l'absence de certaines informations. Par exemple, l'attribut *Fatigue* (de la classe *Well-being*), qui indique l'état de l'utilisateur au moment de l'interaction avec le système, peut être mappé avec la classe *Tiredness*. Notons bien que l'attribut du modèle de contexte *Fatigue* est de type "Boolean".
- **Le mapping indirect :** le troisième type se réfère à certains attributs du modèle de contexte qui peuvent avoir une influence indirecte sur les concepts de l'ontologie. Ceci signifie qu'ils peuvent avoir indirectement un impact sur le contenu du concept du domaine. Dans notre cas, la mauvaise nutrition (la classe *Bad_nutrition*) peut induire une diarrhée (l'attribut *Diarhhea*). Un autre exemple de ce mapping indirect, est l'information concernant la possibilité d'avoir un infarctus chez l'utilisateur (l'attribut *Heart_block*) à cause d'un manque de pratique de sport (la classe *Lack_of_exercice*). Étant donné que la constipation chez une personne peut causer des maux abdominaux, la classe *Abdominal_Pain* est mappée aussi indirectement avec l'attribut *Constipation*. En suivant la même logique de raisonnement et sachant que le manque de gratitude et de joie chez une personne peut causer sa dépression, le concept *Lack_of_gratitude* de l'ontologie du domaine et l'attribut *Depression* appartenant au modèle de contexte sont mappés ensemble. Le dernier mapping indirect dans cet exemple est réalisé entre la classe *Gender* et l'attribut *Caesarean*. De même, la préférence de choisir le mode chirurgical lors de l'accouchement (césarienne), dépend du genre de la personne (homme/femme) ; dans ce cas il n'est valable que pour les femmes.

La Figure 5.11 résume l'ensemble des mappings entre l'ontologie des maladies et les éléments du modèle de contexte.

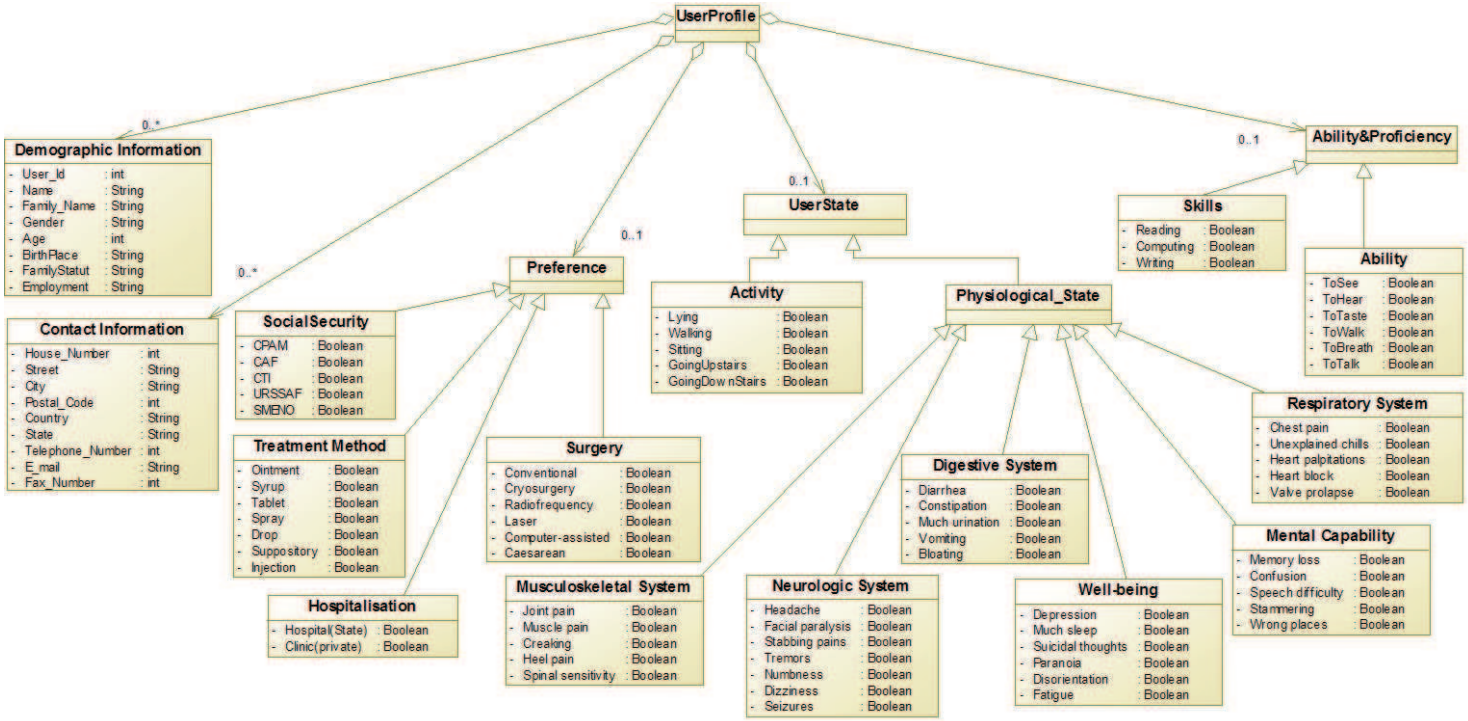


FIGURE 5.10 – Profil utilisateur spécifique au domaine de médecine

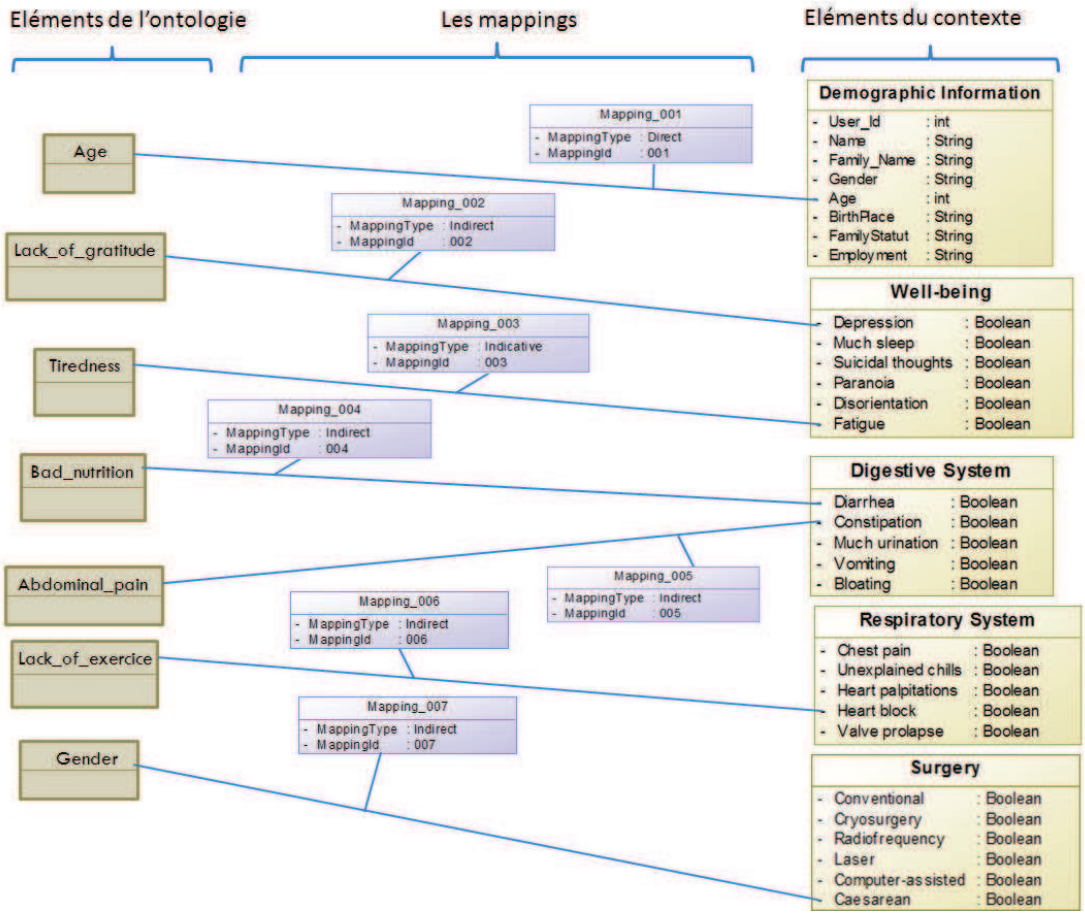


FIGURE 5.11 – Modèle de mapping entre l'ontologie des maladies et le modèle de contexte

Une fois que nous avons élaboré les mappings, entre l'ontologie de domaine et le modèle de contexte, nous pouvons les utiliser dans la définition du modèle des tâches modélisant le système d'assistance médicale. La section suivante présente les étapes suivies pour élaborer ce modèle et la façon selon laquelle nous utilisons ces mappings pour assurer la personnalisation du contenu.

5.2.3.2 Étape 1 : Élaboration du modèle des tâches

En se basant sur la version étendue de BPMN, nous présenterons le modèle des tâches du système précédent (cf. Figure 5.12). Ce modèle est établi en deux étapes : dans la première, le concepteur élabore un modèle des tâches non annoté puis, dans la seconde, il annote ce modèle avec les informations requises afin d'y apporter la personnalisation du contenu.

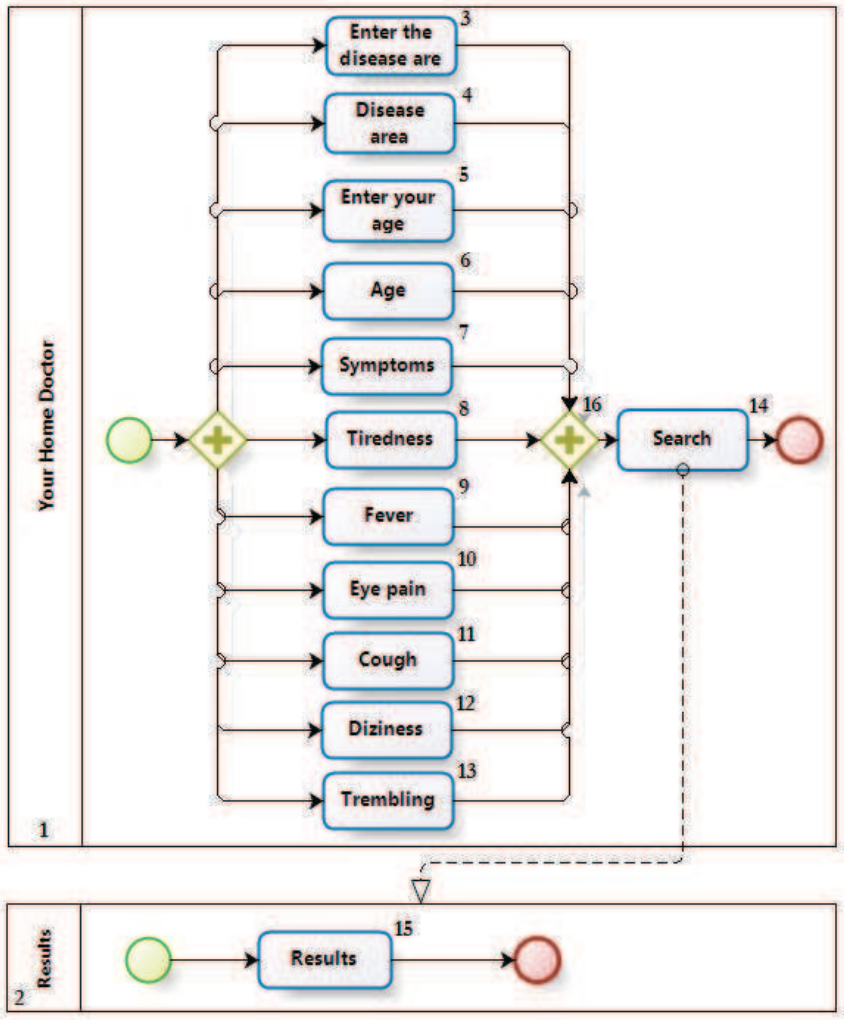


FIGURE 5.12 – Modèle des tâches modélisant le système d'assistance médicale

Afin de créer un modèle des tâches non annoté respectant les règles définies au niveau du chapitre 3, nous concluons que :

- Le système précédent contient deux espaces de travail - "Your Home Doctor" et "Results", et qu'il lui faudra deux *"Pools"*.
- Les tâches de (3) à (13) peuvent avoir lieu simultanément. Pour mettre en œuvre cette idée, nous avons utilisé l'élément *"Parallel Gateway"* lors de la dissociation et lors de l'association des *"sequence flow"*. Après la fin de ces tâches, l'utilisateur peut lancer le processus de recherche des résultats (tâche numéro (14)). L'élément de type *"message flow"* sortant de la tâche numéro (14) et rejoignant le *"Pool"* numéro (2) correspond à un changement d'espace de travail lors du lancement de la recherche.

Au moment de la réception de ce message, le processus inclus dans le *"Pool"* numéro (2) peut commencer. Dans notre cas d'application, il est composé de la tâche (15) permettant l'affichage des résultats de recherche.

5.2.3.3 Étape 2 : Annotation du modèle des tâches

Une fois que nous avons défini le modèle des tâches non annoté, nous devons l'annoter avec des informations pouvant agir sur la personnalisation du contenu, lors de l'exécution. Cette annotation peut être appliquée, si nécessaire, sur chaque *"Pool"*, *"task"*, *"SubProcess"* ou bien *"Group"*. L'idée est d'introduire dans ce modèle des tâches non annoté, l'âge et la fatigue de l'utilisateur comme informations que le système est censé connaitre en fonction du contexte. Grâce à ces informations, le système peut fournir à l'utilisateur un formulaire pré-rempli. En plus, pendant l'annotation du modèle des tâches, nous devons tenir compte des requêtes à enrichir automatiquement, à l'exécution. Dans cet exemple, supposant que nous voulons que le système prenne en compte, pendant la recherche, la possibilité de souffrance de l'utilisateur de maux de ventre et la possibilité qu'il soit dépressif.

En suivant les règles d'association présentées précédemment (cf. chapitre 3, §3.3), le Tableau 5.4 résume l'annotation de notre modèle BPMN avec des éléments du modèle d'interaction.

Table 5.4 – Annotation du modèle des tâches par les éléments d'interaction (cas du système d'assistance médicale)

Identifiant de l'élément BPMN	**Type de l'élément BPMN**	**Élément d'interaction associé**
1, 2	Pool	UIGroup
3, 7, 5	SystemTask	UIFieldStatic
4, 6	UserTask	UIFieldInManual
8, 9, 10, 11, 12, 13	UserTask	UIFieldOneChoice
14	UserTask	UIFieldAction
15	SystemTask	UIFieldOut

Après avoir associer les éléments d'interaction aux différents composants du modèle des tâches, et pour appliquer la totalité des règles d'annotation du modèle BPMN, présentées dans le chapitre 3, §3.3), on peut procéder comme suit :

- Sur la base de la règle 1 et des mappings existants, nous associons à la tâche (6), le concept de l'ontologie *Age* via l'attribut *OntologyElementName* et nous choisissons l'option directe (pour l'attribut *mapping type*). Ceci signifie que la valeur du champ

de saisie "Age" correspond à la valeur de l'élément de contexte mappé directement à cette tâche, dans ce cas, l'attribut *Age*.

- D'après la règle 2, nous associons à la tâche (8) le concept *Tiredness* appartenant à l'ontologie. Ensuite, nous assignons au mapping le type "Indicative", afin de montrer que la sélection de cette alternative dépend de l'état de l'utilisateur (fatigué ou pas).
- En fonction de la règle 3, nous associons à la tâche numéro (15) le concept recherché *Disease* (via l'attribut *OntologyElementName*) et ensuite nous mentionnons les paramètres qui doivent être pris en compte implicitement pendant la recherche, les classes *Abdominal_pain* et *Lack_of_gratitude*.

La Figure 5.13 résume l'ensemble des annotations apportées aux éléments du modèle des tâches, afin de prendre en compte la personnalisation du contenu.

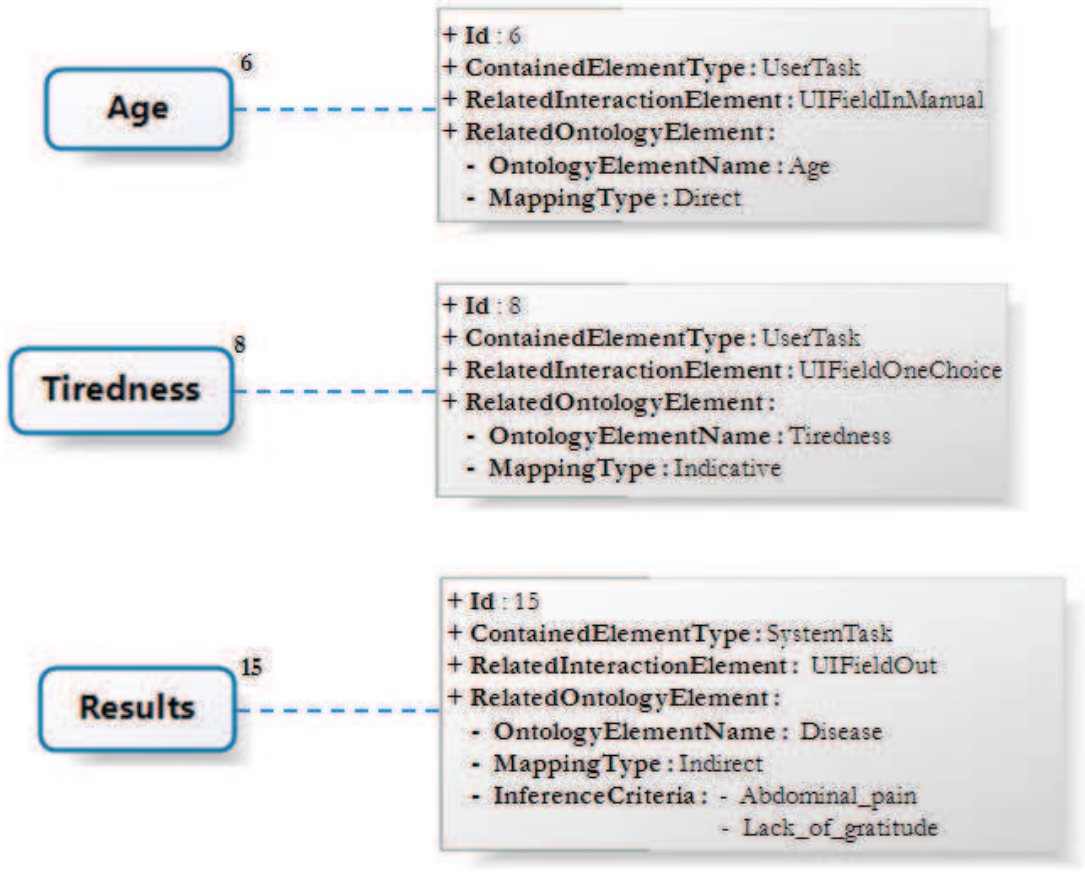


FIGURE 5.13 – Annotation des éléments du modèle des tâches (cas du système d'assistance médicale)

Une fois le modèle des tâches annoté est défini, le concepteur peut lancer le processus de transformation de modèles. Dans la prochaine section, nous allons décrire le code UIML généré pour les niveaux PIM et PSM.

5.2.4 Génération du code UIML

Comme indiqué précédemment dans le chapitre (3) et le premier cas d'étude, le code UIML généré met en œuvre la personnalisation du contenu sur la base de deux méthodes : le remplissage automatique des formulaires et l'enrichissement des requêtes.

Le remplissage automatique des formulaires est effectué pour tous les champs qui ont été dotés d'un mapping direct ou indicatif dans le modèle des tâches. Dans cet exemple, pour le mapping direct, la propriété `g:text` qui contient le contenu de l'élément numéro 6, est remplie automatiquement avec la valeur de l'attribut *Age,* déduite de l'instance du modèle de contexte, par l'appel de la méthode `006GetValueFromContext`. Le code UIML généré montre la manière avec laquelle le champ indiquant l'age de l'utilisateur va être rempli par la valeur de l'attribut *Age* (cf. Figure 5.14). De la même manière, nous traitons les concepts de l'ontologie dotés des mapping indicatifs.

Afin de décider de la sélection ou non de l'élément concerné, en fonction des préférences de l'utilisateur, la valeur de l'élément de contexte (`true / false`) est vérifiée dans la partie `condition` de UIML (via la propriété `g:selected`). La partie du code se trouvant dans la Figure 5.14 est générée pour traiter le cas de fatigue, où la bonne alternative sera pré-sélectionnée en se basant sur l'état de l'utilisateur.

Pour les mappings indirects, la transformation génère au niveau PIM la partie `call` de UIML qui représente une abstraction de tout type d'invocation de méthode externe. Cette méthode est équivalente à une requête de recherche, dans laquelle la première balise présente l'élément recherché (dans notre cas *Disease*) et la suite des balises présentent les paramètres à prendre en compte lors de cette recherche. Comme nous sommes intéressés de fournir un contenu personnalisé pour l'utilisateur, cette requête a été enrichie par l'inclusion automatique d'autres éléments de l'ontologie. Cet enrichissement s'effectue en respectant les règles présentées dans le chapitre 4. Le Tableau 5.5 résume la manière avec laquelle l'ensemble des relations et des axiomes de l'ontologie des maladies a été exploité pour appliquer ces règles.

TABLE 5.5 – Enrichissement de la requête en fonction des axiomes de l'ontologie des maladies

Caractéristique de la relation	**Comment exploiter l'axiome ?**
FunctionalProperty	Pour rechercher l'élément *Disease*, le concepteur a défini la classe *Abdominal_pain* comme un *InferenceCriteria*. Parmi les éléments faisant partie du chemin reliant ces deux classes, la transformation détecte la classe *Patient*. Cette classe possède une propriété fonctionnelle (*has_gender*) la reliant à la classe *Gender*. Cette dernière est dotée d'un mapping indirect. Ainsi, la classe *Gender*, sera automatiquement ajoutée à *InferenceCriteria* pendant la recherche du *Disease*.
SymmetricProperty	Pour rechercher l'élément *Disease*, le concepteur a défini les classes *Abdominal_pain* et *Lack_of_gratitude*. En vérifiant les propriétés reliant ces classes, la seule propriété symétrique trouvée est *is_derived_from*. Cette dernière relie la classe *Causes* à elle même. Par conséquent, dans ce cas, aucune nouvelle classe ne sera ajoutée à *InferenceCriteria*.
TransitiveProperty	Pour rechercher l'élément *Disease*, le concepteur a défini la classe *Lack_of_gratitude* comme *InferenceCriteria*. Comme la classe *Bad_nutrition* est dotée d'un mapping indirect et comme la classe *Causes* est une super-classe pour *Lack_of_gratitude* et *Bad_nutrition*, ayant une propriété (*is_derived_from*) transitive, alors *Bad_nutrition* sera automatiquement ajoutée à *InferenceCriteria* pendant la recherche de *Disease*. De même, la classe *Lack_of_exercice* est dotée d'un mapping indirect et elle est une sous-classe de *Causes*. Par conséquent, *Lack_of_exercice* est également ajoutée à *InferenceCriteria* pendant la recherche de *Disease*.

Le code UIML généré au niveau PIM, dans le cas des mappings indirects comprend une méthode appelée `Get-Element` qui est appelée via l'instruction `call`. Ses paramètres sont l'élément recherché (*Disease*) suivi des paramètres que le système devrait prendre en considération lors du processus de recherche (*Abdominal_pain*, *Lack_of_gratitude*, *Gender*, *Bad_nutrition*, *Lack_of_exercice*) (cf. Figure 5.14).

```
<UIML:Structure>
...
<part class="G:TopContainer" id="1">
   ...
  <part class="G:TextField" id="3"/>
  <part class="G:CheckBoxButton" id="8"/>
   ...
</part>
...
</UIML:Structure>

<UIML:Behavior id="Main Behavior">
 <rule id="Rule6">
  <condition>
   <op name="Equal">
    <variable name="6isactivated"/>
    <constant value="true"/>
   </op>
  </condition>
  <action>
   <whenTrue>
    <property name="g:text" partName="6">
     <call componentId="6Context"  methodId="6GetValueFromContext">
        <param name="Age"/>
     </call>
    </property>
    <property name="g:visible" partName="6">
       <constant value="true"/>
    </property>
    <variable name="16isactivated" value="true"/>
   </whenTrue>
  </action>
 </rule>

 <rule id="Rule15">
  <condition>
   <op name="Equal">
    <variable name="15isactivated"/>
    <constant value="true"/>
   </op>
  </condition>
  <action>
   <whenTrue>
    <property name="g:text"partName="15">
     <call componentId="15Context" methodId="15Get-Element">
      <param name="Disease"/>
      <param name="Abdominal_pain"/>
      <param name="Lack_of_gratitude"/>
      <param name="Gender"/>
      <param name="Bad_nutrition"/>
      <param name="Lack_of_exercice"/>
     </call>
    </property>
    ...
   </whenTrue>
  </action>
 </rule>

</UIML:Behavior>
```

La partie structure

La partie Behavior (cas d'un mapping direct)

La partie Behavior (cas d'un mapping indirect)

FIGURE 5.14 – Exemple de code UIML généré lors du passage du CIM vers PIM

La Figure 5.15(a) illustre un exemple d'une IHM correspondante au système étudié. La personnalisation du contenu est mise en œuvre à travers deux aspects. En fait, l'âge de l'utilisateur et son état (la fatigue) sont automatiquement complétés par le système. Un autre exemple de personnalisation du contenu est présenté dans la Figure 5.15 (b). En effet, le système prend en compte non seulement des éléments explicites de la requête de l'utilisateur (âge, zone atteinte et symptômes), mais inclut automatiquement d'autres données implicites déduites du contexte afin de fournir un résultat plus personnalisé. Dans ce cas, étant donné la connaissance du système de l'état de l'utilisateur (maux de ventre), la requête initiale sera étendue. Ainsi, le résultat retournera uniquement les maladies causant de la fièvre et des maux de ventre.

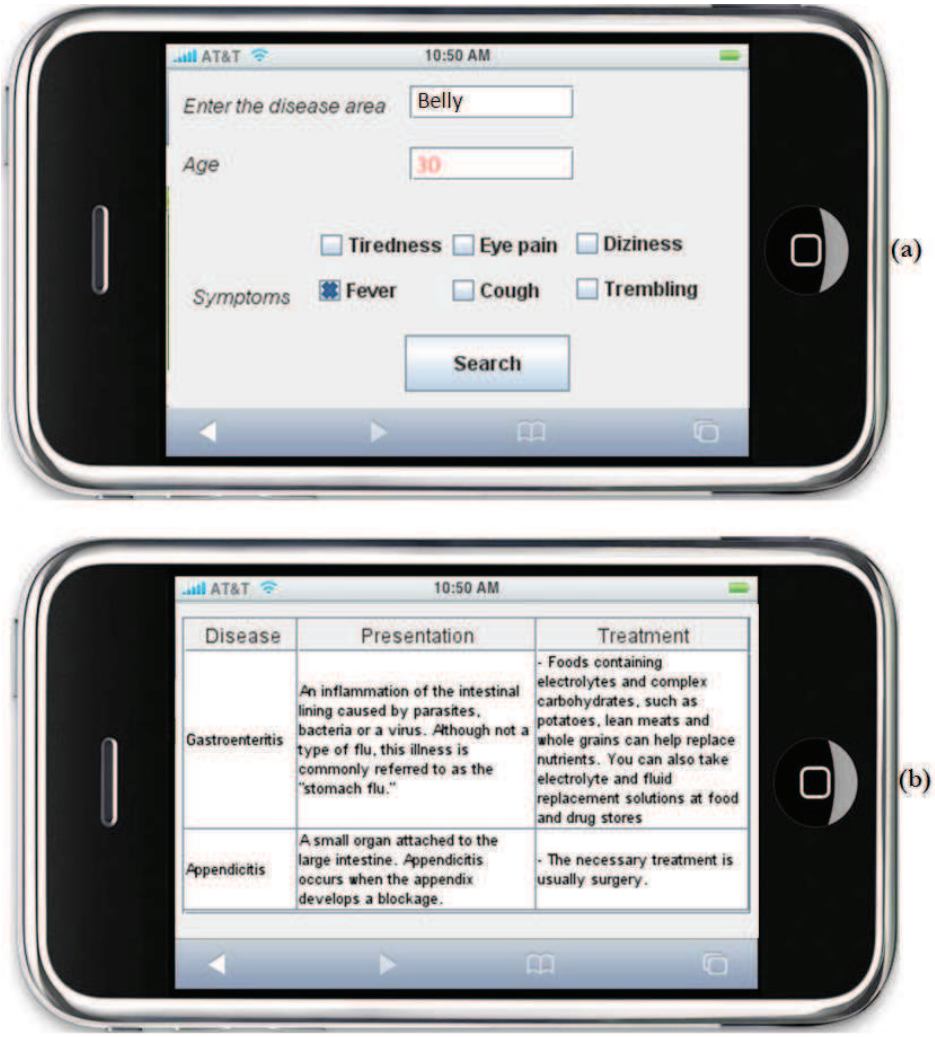


FIGURE 5.15 – Exemples d'IHM générées pour le système d'assistance médicale

Conclusion

A travers ce chapitre, nous avons pu souligner et mettre en œuvre une des principales caractéristiques de notre approche, à savoir sa généricité et par conséquent son indépendance du domaine d'application. Cette généricité a été validée par deux études de cas appartenant à deux domaines différents : l'une traite le domaine de transport en commun (Bacha *et al.*, 2011b) et l'autre le domaine médical (Bacha *et al.*, 2011c, 2013).

Conclusion générale

Les travaux de recherche présentés dans ce mémoire se situent dans les thématiques de la génération des applications interactives et de la personnalisation du contenu. L'objectif de cette thèse est de proposer une approche de type MDA, indépendante du domaine d'application, permettant la conception et la génération semi-automatique des applications interactives à contenus personnalisés, compte tenu des informations sur le contexte d'utilisation et l'ontologie du domaine. Cette approche met en œuvre deux méthodes de personnalisation, à savoir le remplissage automatique des formulaires et l'enrichissement des requêtes. Pour atteindre cet objectif, nous avons développé la solution technique qui permet la conception, la transformation des modèles ainsi que la génération de l'IHM finale.

Évaluation de l'approche

Contributions principales de la thèse

Cette thèse nous a permis de réaliser les contributions suivantes :

- Proposer une approche de type MDA, permettant de prendre en compte la personnalisation du contenu lors de la conception et la génération des IHM. Cette approche met en œuvre deux méthodes de personnalisation du contenu : le remplissage automatique des formulaires et l'enrichissement des requêtes (Bacha *et al.*, 2011e) . Notre approche est indépendante du domaine, et sa généricité a été validée par deux études de cas appartenant à deux domaines différents : l'une traite le domaine de transport en commun (Bacha *et al.*, 2011b) et l'autre le domaine médical (Bacha *et al.*, 2011c, 2013). À travers ces études, nous avons détaillé les différentes étapes de modélisation et de génération des deux applications.
- Concevoir un modèle générique de contexte représentant le cœur de notre approche. À cette fin, nous avons effectué une revue détaillée de la littérature traitant 18 propositions de modélisation de contexte. Au cours de cette étude, nous avons analysé tous les concepts et les propriétés des modèles de contexte proposés. Ensuite, nous avons classé ces éléments par catégories de concepts en fonction de leurs sens sémantiques. Finalement, nous avons organisé ces catégories autour des trois dimensions du contexte, à savoir l'utilisateur, la plateforme et l'environnement (Bacha *et al.*, 2011e).
- Présenter un modèle de mapping qui permet de faire la correspondance entre les éléments du modèle de contexte et ceux de l'ontologie de domaine représentant l'ensemble des données de l'application. Nous avons proposé trois types de mappings (direct, indicatif, indirect) permettant de définir la manière avec laquelle une information du domaine peut être influencée par un élément du contexte (Bacha *et al.*, 2011a,e). Notons qu'un mapping peut être réutilisé dans la conception de plusieurs applications appartenant au même domaine.
- Développer un modèle des tâches étendant le formalisme BPMN afin de soutenir la personnalisation du contenu. Les extensions se concentrent principalement sur la possibilité d'annoter le modèle des tâches, par les informations issues de l'ensemble des modèles

de personnalisation proposés dans l'approche. Notre objectif était d'identifier, dès la phase de conception des IHM, les informations qui doivent être personnalisées pour un domaine spécifique compte tenu du contexte d'utilisation.
– Exploiter l'ontologie du domaine pendant la conception et la génération des IHM à contenus personnalisées. À notre connaissance, notre approche est la première qui exploite la totalité des éléments de l'ontologie (concepts, relations et axiomes) pour mettre en œuvre la personnalisation du contenu dans une approche basée sur les modèles (Marcal de Oliveira *et al.*, 2013).
– Définir une logique de transformation du modèle BPMN vers le langage UIML générique et développer les règles ATL permettant de mettre en œuvre l'ensemble de ces ces transformations. Cette transformation conserve la dynamique du comportement de l'IHM, au moment de l'interaction avec l'utilisateur, ainsi que le passage du flux d'information, entre les tâches interactives et non interactives (Bacha *et al.*, 2011d).
– Proposer un atelier logiciel, permettant de créer, de manipuler et de transformer l'ensemble des modèles conceptuels de notre approche. Cet atelier a permis de valider notre approche et de la mettre en pratique.

Comparaison de notre approche par rapport aux autres

En se basant sur les critères de comparaison présentés dans la section 2.1, nous présenterons un tableau comparatif permettant de positionner notre proposition par rapport aux autres (voir table C.1) :
– *Conformité à MDA* - Contrairement à la plupart des approches déclaratives étudiées dans le chapitre 2, notre approche est conforme à MDA, faisant partie de la même lignée que celles proposées par (Limbourg *et al.*, 2005; Brossard, 2008; Bouchelligua *et al.*, 2010). Ce nombre limité d'approche de type MDA est expliqué par la difficulté de découpler les aspects liés à la plateforme depuis les plus hauts niveaux de conception.
– *Transformation des modèles* - Certaines approches n'ont pas implémenté de règles de transformation pour les modèles qu'ils proposent et d'autres n'ont pas fourni suffisamment de détails sur la manière avec laquelle ils traitent leurs modèles. Cependant, dans le cas des approches où les règles de transformations ont été implémentées, la transformation des modèles en utilisant les langages spécifiques de transformation reste la méthode la plus valide en IDM. Dans ce cadre, nous avons présenté la logique de transformation de nos modèles ainsi que l'ensemble des règles ATL développées pour mettre en œuvre l'ensemble de ces transformations.
– *Modélisation du contexte* - Afin d'implémenter leur démarche d'adaptation, certains auteurs ont développé leurs propres modèles de contexte à différents niveaux d'abstraction mais la plupart des auteurs n'ont pas fourni de modèles de contexte. Ayant des besoins et des objectifs différents, certaines approches se sont basées sur la totalité des éléments du contexte et d'autres se sont contentées d'utiliser une partie de ces éléments. Concluant que les modèles du contexte proposés étaient, soit très génériques d'une manière à rendre difficile la personnalisation du contenu, soit très spécifiques pour un domaine particu-

lier, nous avons proposé notre modèle de contexte indépendant du domaine, qui couvre toutes les dimensions.
- *Cible d'adaptation* - Dans les approches présentées, les auteurs visent majoritairement l'adaptation de la présentation de l'interface et à notre connaissance, l'unique proposition qui s'intéressait à la personnalisation du contenu était PERCOMOM (Brossard, 2008). Contrairement à PERCOMOM qui se contente d'utiliser uniquement les concepts de l'ontologie, nous exploitons la totalité des éléments de l'ontologie (concepts, relations et axiomes) pendant la conception et la génération des IHM à contenus personnalisés.
- *Moment d'adaptation* - Bien que notre approche prend en compte la personnalisation du contenu pendant le runtime, l'adaptation du contenant est effectuée pendant le design time.
- *Outillage proposé* - Certaines approches sont orientées implémentation et elles proposent des outils qui permettent de spécifier leurs modèles et leurs mécanismes de transformation. D'autres sont plutôt orientées conception, où les auteurs se contentent de proposer uniquement leurs méthodologies théoriques. Dans notre proposition, nous avons proposé un atelier logiciel permettant la modélisation et la génération des IHM à contenus personnalisés.

Limites et perspectives

Si notre approche présente un certain nombre d'avancées en matière de conception et de générations des applications interactives à contenus personnalisés, elle présente également, un certain nombre de points d'améliorations qui font partie de nos principales perspectives de recherche :
- *La nécessité d'une ontologie de domaine* - En effet, les approches se basant sur IDM ou MDA considèrent, généralement, un modèle de domaine défini spécifiquement pour le système à développer. Nous avons choisi d'utiliser l'ontologie de domaine vu qu'elle permet de définir le domaine indépendamment du système et est donc susceptible d'être réutilisée pour le développement de diverses applications du même domaine. Néanmoins, nous payons l'effort d'avoir une ontologie.
- *Nécessité de définir le mapping entre l'ontologie du domaine et modèle de contexte* - Cette correspondance nécessite une connaissance approfondie du domaine d'application afin de choisir les éléments de l'ontologie et ceux du modèle de contexte, qui doivent être mappés ensemble. Cependant, une fois qu'ils ont été définis, les mappings peuvent être utilisés dans plusieurs applications. Par conséquent, l'utilisation de notre approche se justifie plus dans le cas où nous sommes ramenés à développer plusieurs applications pour le même domaine d'application. En plus, à l'état actuel, la réalisation des mappings se fait manuellement mais une automatisation de ce processus sera bénéfique en termes de gain du temps.
- *Dépendance de UIML* - Notre approche génère du code UIML générique qui doit être traduit par la suite pour une plateforme spécifique en utilisant des interpréteurs du code UIML ou bien des générateurs du code source. Pour faire face à la traduction de la

dynamique de l'IHM, présentée dans le modèle des tâches, ainsi qu'à la personnalisation du contenu, nous avons utilisé la notion de *variable* qui est fournie uniquement dans la version 4.0 de UIML. Néanmoins, il n'existe pas plusieurs outils qui adoptent cette dernière version de UIML, ce qui limite notre approche lors du choix du générateur de l'IHM finale.

– *Nécessité du codage pour les mappings indirects* - Dans le cas des mappings directs et indicatifs, les transformations incluent directement des informations du contexte qui doivent être prises en considération. Cependant pour les mappings indirects et afin d'exprimer la requêtes de recherche d'information, nous appelons les méthodes avec les paramètres convenables. Dans le cas d'une interprétation du code UIML généré, ces méthodes doivent être codées préalablement. Dans le cas d'une génération du code source à partir du code UIML, le codage de ces méthodes doit être réalisé par la suite.
– *Développement d'un atelier logiciel intégré* - Pour mettre en pratique notre approche, nous nous sommes basés sur une chaine d'outils, permettant de générer des IHM à contenus personnalisés. En revanche, afin de mieux appuyer le concepteur pendant le processus de conception et de génération des IHM, il est apprécié de lui fournir un environnement de développement intégré et compact.
– *Expérimentation de l'approche dans plusieurs domaines d'application* - A son état actuel, notre approche a été étudiée et validée sur deux domaines d'application : le domaine de transport en commun et le domaine médical. Néanmoins, l'exploitation d'autres domaines garantira mieux sa généricité et son indépendance de tout domaine d'application.

TABLE C.1 : Comparaison de notre approche avec celles présentées dans le chapitre 2

		Notre approche	ArtStudio	TERESA	SUPPLE	UsiXML	Dynamo-Aid	PERCOMOM	(Sottet et al., 2007)	(Hachani et al., 2009)	(Bouchelligua et al., 2010)
Conformité à MDA	Conforme à MDA	✓				✓		✓			✓
	Non conforme à MDA		✓	✓	✓		✓		✓	✓	
Transformation des modèles	Basée sur les langages de transformation	✓							✓	✓	✓
	Basée sur la théorie des graphes					✓					
	Basée sur l'optimisation mathématique				✓						
	Technique non fournie		✓	✓			✓				
	Pas de transformation développée							✓			
Modélisation du contexte	Modèle total	✓				✓			✓	✓	✓
	Modèle partiel		✓		✓						
	Modèle non fourni			✓			✓				
	Modèle non développé							✓			
Cible d'adaptation	Contenu	✓						✓			
	Contenant		✓	✓	✓	✓	✓		✓	✓	✓
Moment d'adaptation	Runtime	✓		✓	✓		✓				
	Design time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Outillage proposé	Outillage complet	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	Outils de modélisation uniquement							✓			
	Mécanisme de transformation uniquement								✓	✓	✓

Troisième partie

Annexes

Annexe A1

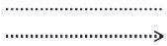
Les modèles des tâches

BPMN

Un processus métier est un ensemble de procédures ou des activités qui sont liées, afin de réaliser un objectif, dans le cadre d'une structure organisationnelle définissant les rôles et les relations fonctionnelles. Dans la notation BPMN 2.0 , un processus est représenté sous la forme (1) d'un graphe d'éléments qui peuvent être des activités, des tâches, des évènements et des passerelles et (2) de flux séquentiel qui décrit sémantiquement l'ordre dans lequel ce graphe d'éléments doit être réalisé. Les processus peuvent être modélisés selon plusieurs niveaux de décomposition hiérarchique. Le Tableau A1.1 présente une liste des éléments de base proposés par BPMN 2.0.

Table A1.1: Les éléments de base de la notation BPMN

Élément	**Notation**	**Signification**
Tâche (*Task*)		Représente tout travail qui est accompli à l'intérieur d'un processus. Une activité consomme du temps, une ou plusieurs ressources, requiert un ou plusieurs objets et produit un ou plusieurs objets. Les activités peuvent représenter plusieurs niveaux de détails. Lorsque les activités sont combinées ensemble, elles forment un Sous-processus.
Sous-processus (*Sub-process*)		Les activités peuvent représenter plusieurs niveaux de détails. Lorsque les activités sont combinées ensemble, elles forment un sous-processus
Évènement (*event*)		Représente quelque chose qui se produit dans le cours normal de la réalisation d'un processus. Il existe trois types d'évènement : déclencheur, résultant et intermédiaire.
Passerelle (*gateway*)		Utilisée pour contrôler les divergences et les convergences que l'on retrouve au niveau des flux séquentiels d'activités ou de tâches dans un processus. Il existe plusieurs types de passerelle : exclusive, inclusive, parallèle, complexe et évènement.

Flux séquentiel (*sequence flow*)		Représente l'ordre dans lequel les activités ou les tâches doivent être réalisées au sein d'un Processus
Flux de messages (*message flow*)		Représente les échanges de messages, c'est-à-dire les émissions et les réceptions de messages entre deux Participants.
Association (*association*)		Utilisée afin d'illustrer les relations entre les éléments graphiques du BPMN 2.0 tels que les annotations, les données, les messages et les objets du flux.
Participant (*pool*)		Représente soit une entité externe (ex : une autre entreprise) et/ou une Entité qui joue un rôle particulier (ex : un acheteur, un vendeur, etc.) qui sont les participants d'une collaboration.
Couloir (*lane*)		Représente la division d'un processus en sous-ensemble d'activités ou de tâches.
Objet de données (*data object*)		Fournit de l'information sur les activités ou les tâches qui doivent être réalisées.
Message (*message*)		Utilisé pour décrire le contenu d'une communication qui se déroule entre deux participants.
Regroupement (*group*)		Permet de regrouper des éléments graphiques qui appartiennent à une même Catégorie. Le Regroupement n'a pas de conséquence sur le flux séquentiel.
Texte (*text annotation*)		Descriptif permet au modélisateur d'ajouter des informations supplémentaires à son modèle.

CTT

CTT (ConcurTask Trees) provient du : Human Computer Interaction Group - ISTI (Pise, Italie). Il s'agit d'un formalisme graphique, spécialement conçu pour la modélisation des IHM. Doté d'un éditeur graphique CTTE (Mori *et al.*, 2002) et d'un générateur d'IHM TERESA (Mori *et al.*, 2004), CTT propose quatre catégories de tâches détaillées dans le Tableau A1.2 :

TABLE A1.2 – Les catégories de tâches CTT

Types de tâche	**Notation**	**Signification**
Tâche de l'utilisateur		Tâche cognitive, par ex. choisir une méthode de résolution de problème.
Tâche de l'application		Tâche où seul le système informatique intervient, par ex. produire le résultat faisant suite à une requête.
Tâche d'interaction		Actions de l'utilisateur sur le système avec possibilité d'un retour immédiat.
Tâche abstraite		Tâches de plus haut niveau se décomposant en sous-tâches pouvant appartenir à l'une des quatre catégories.

Ces tâches peuvent être reliées par dix opérateurs temporels tels que présentés dans le Tableau A1.3 :

TABLE A1.3 – Les catégories de tâches CTT

Opérateur	**Notation**	**Signification**
Concurrence (*Concurrent*)	T1 ||| T2	Les deux tâches peuvent être effectuées en parallèle sans aucune contrainte
Concurrence avec échange d'information (*Concurrent with info exchange*)	T1 | [] T2	Les 2 tâches doivent se synchroniser pour échanger de l'information.
Choix (*Choice*)	T1 [] T2	Les deux tâches sont alternatives ; le début de l'exécution de l'une empêche l'autre de s'exécuter tant que la première n'est pas terminée.
Indépendance d'ordre (*Order independency*)	T1|=|T2	Les deux tâches doivent être exécutées dans un ordre quelconque (T1 puis T2, ou T2 puis T1).
Désactivation (*Disabling*)	T1⊏≻ T2	La tâche T1 est définitivement stoppée dès que la tâche T2 commence à s'exécuter.
Activation (*Enabling*)	T1≻≻ T2	Lorsque la tâche T1 est terminée, la tâche T2 commence à s'exécuter.
Activation avec échange d'information (*Enabling with info exchange*)	T1[] ≻≻T2	En plus, la tâche T1 transmet des informations à la tâche T2.
Suspension-reprise (*Suspend/Resume*)	T1|≻T2	La tâche T2 peut suspendre l'exécution de la tâche T1. Lorsqu'elle est terminée, la tâche T1 reprend son exécution là où elle a été interrompue.
Iteration (*Set/Unset iterative task*)	T*	La tâche est exécutée en boucle jusqu'à ce qu'elle soit désactivée par une autre tâche.
Tâche optionnelle (*Set/Unset optional task*)	[*T*]	L'exécution de la tâche est facultative.

La Figure A1.1 représente un exemple d'un scénario de réservation d'une chambre d'hôtel établit en CTT, en utilisant l'environnement CTTE.

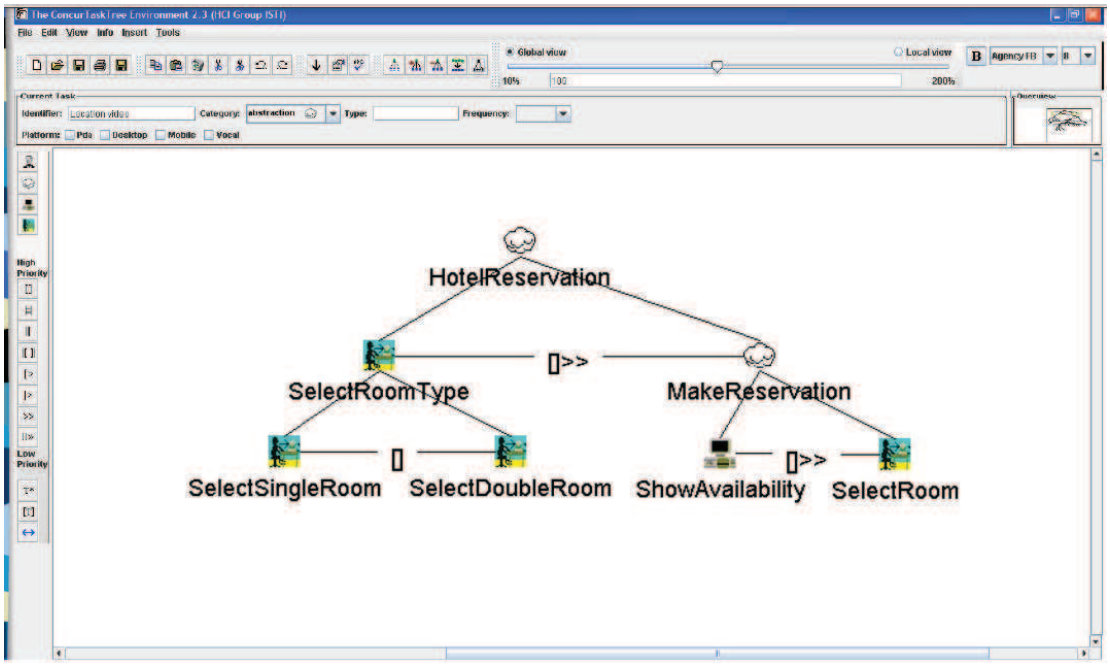


FIGURE A1.1 – Exemple d'un scénario en CTT établi via CTTE

Annexe A2

Les métamodèles Ecore dans notre approche

Métamodèle de BPMN

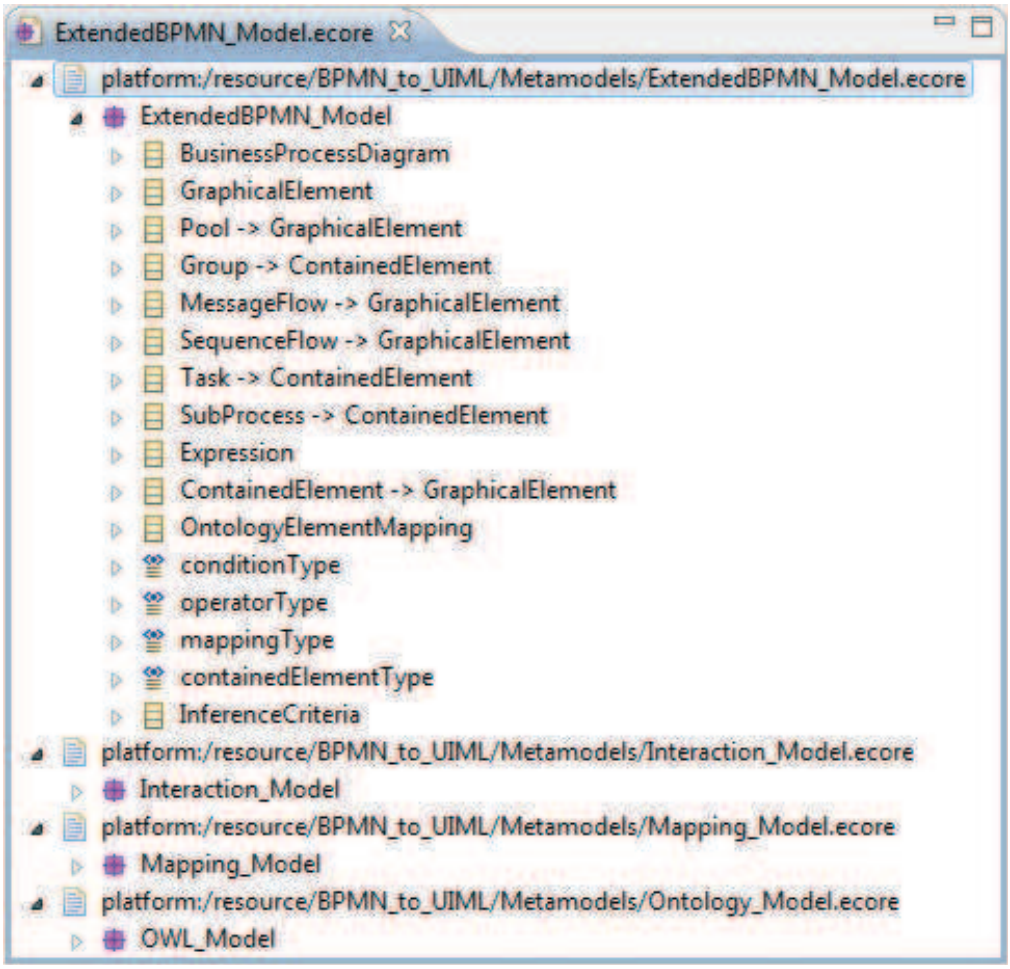


FIGURE A2.1 – Le métamodèle Ecore du BPMN annoté

Métamodèle d'interaction

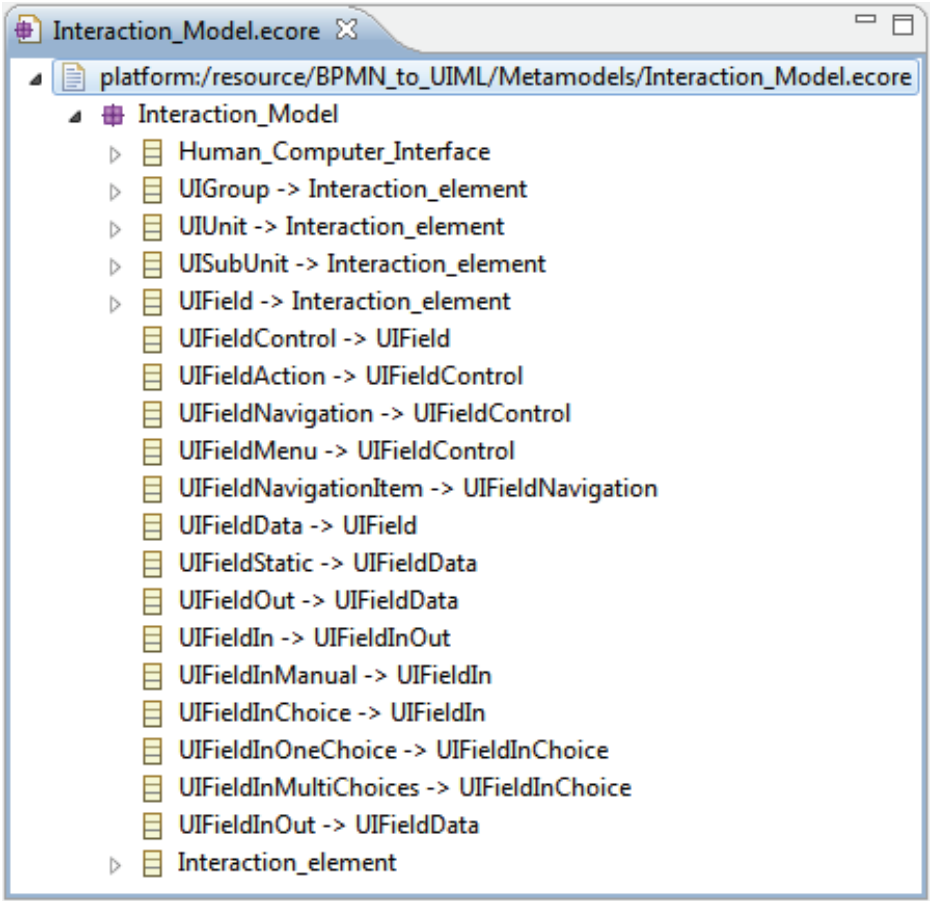


FIGURE A2.2 – Le métamodèle Ecore d'interaction

Métamodèle de mapping

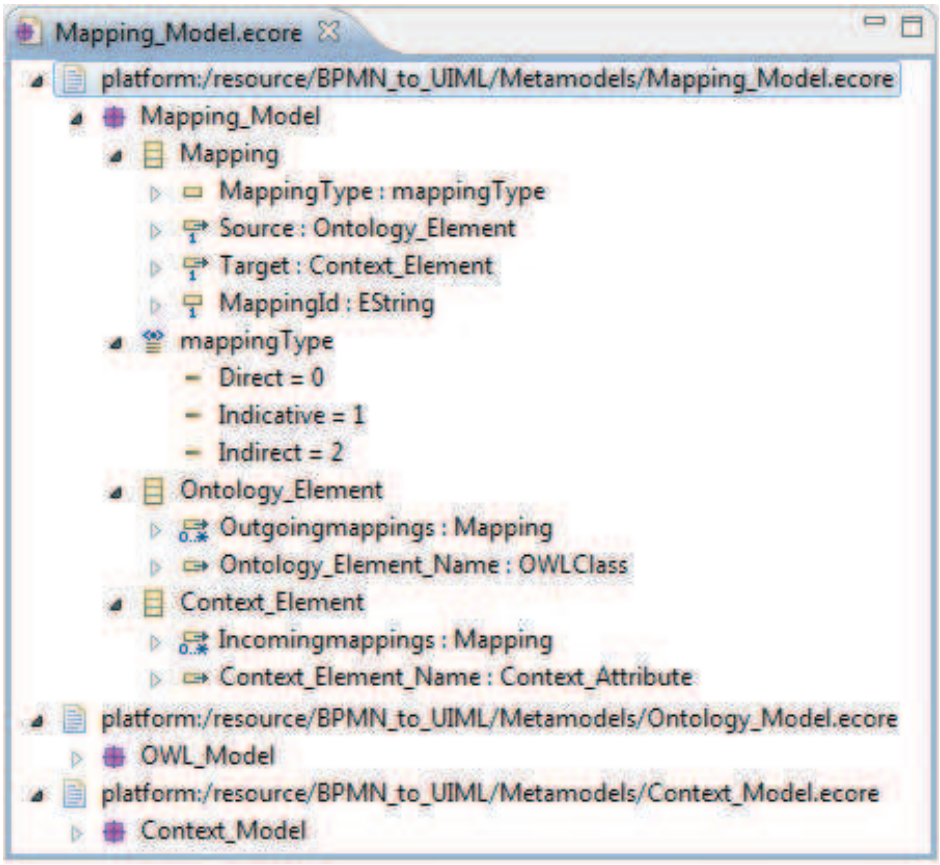


FIGURE A2.3 – Le métamodèle Ecore du mapping

Métamodèle de l'ontologie

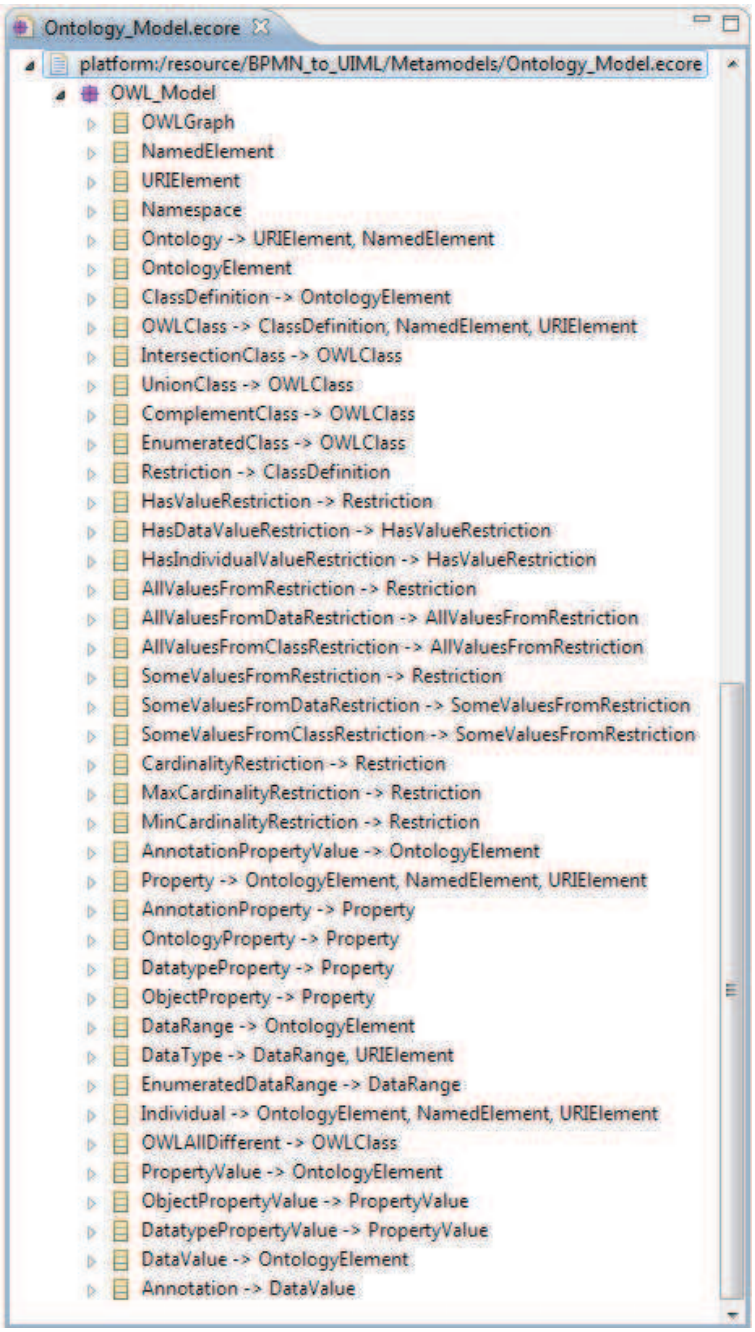


FIGURE A2.4 – Le métamodèle Ecore de l'ontologie

Métamodèle du contexte

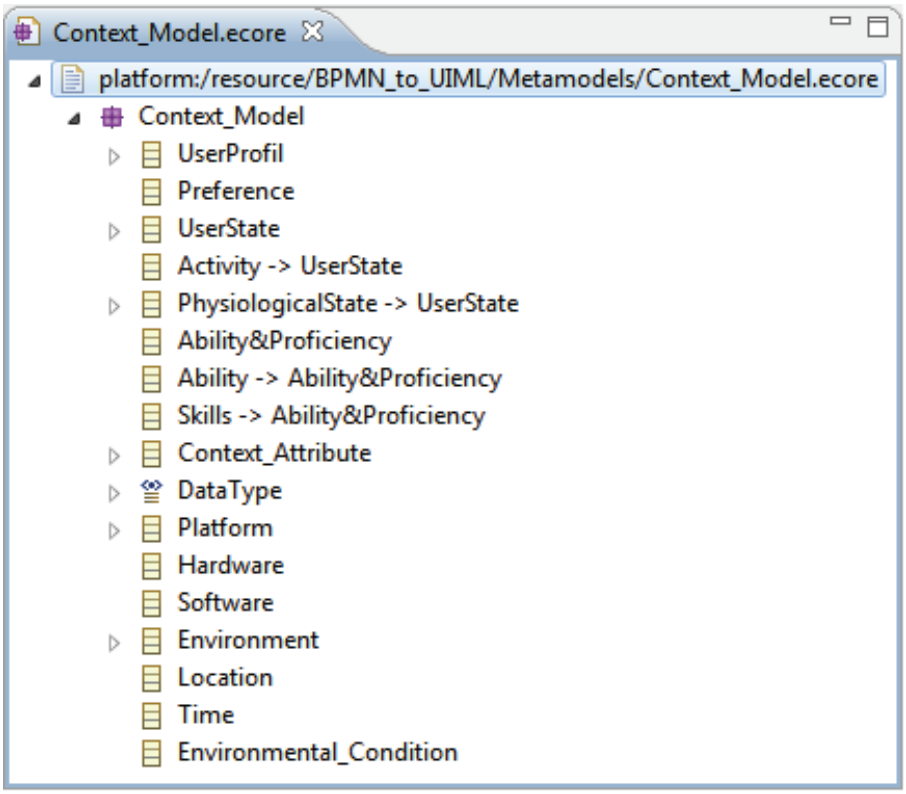


FIGURE A2.5 – Le métamodèle Ecore du contexte

Métamodèle de UIML

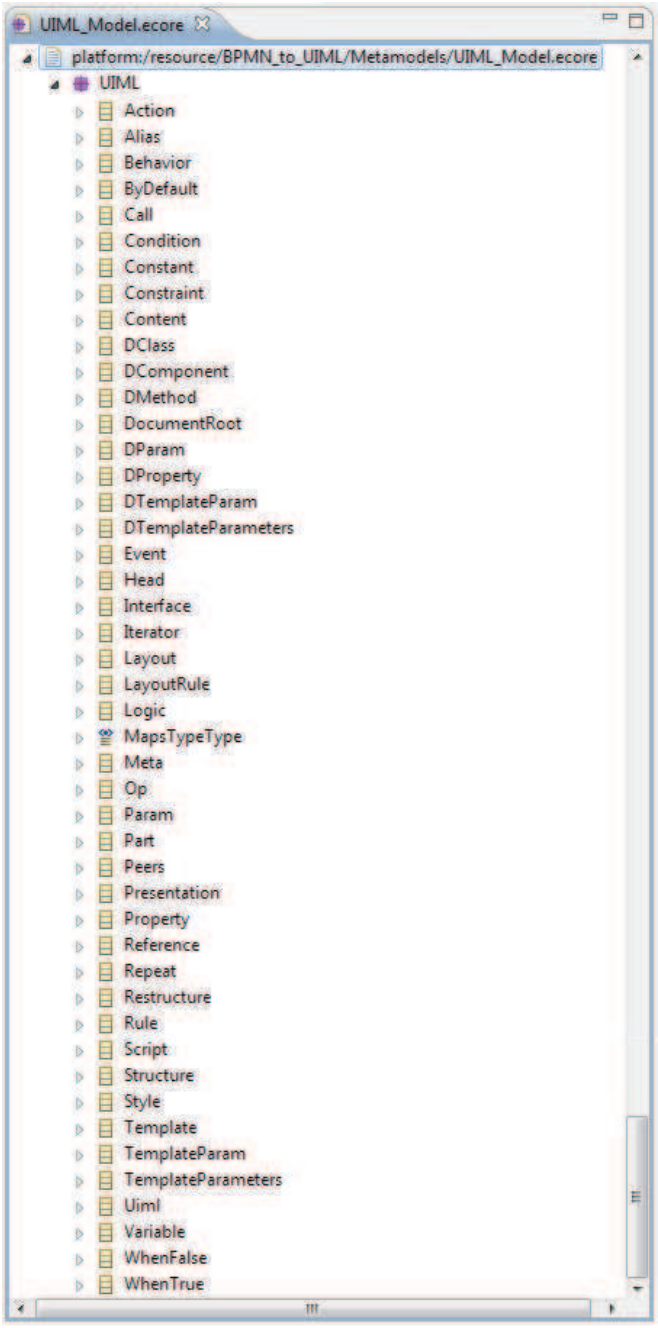


FIGURE A2.6 – Le métamodèle Ecore de UIML

Bibliographie

Abbas, K. (2008). *Système d'accès personnalisé à l'information : application au domaine médical*. Thèse de doctorat, Institut National des Sciences Appliquées de Lyon.

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M. et Shuster, J. E. (1999). UIML : an appliance-independent XML user interface language. *Computer Networks*, 31:1695–1708.

Ajila, S. A., Petriu, D. et Motshegwa, P. (2010). Using model transformation semantics for aspect composition. *2012 IEEE Sixth International Conference on Semantic Computing*, 0:325–332.

Ali, M. F. et Abrams, M. (2001). Simplifying construction of multi-platform user interfaces using uiml. *In European Conference UIML*.

Ali, M. F., Pérez-Quiñones, M. A., Abrams, M. et Shell, E. (2002). Building multi-platform user interfaces with uiml. *In* Kolski, C. et Vanderdonckt, J., éditeurs : *CADUI*, pages 255–266. Kluwer.

Ali, R., Dalpiaz, F. et Giorgini, P. (2008). Modeling and analyzing variability for mobile information systems. *In* Gervasi, O., Murgante, B., Laganà, A., Taniar, D., Mun, Y. et Gavrilova, M. L., éditeurs : *ICCSA (2)*, volume 5073 de *Lecture Notes in Computer Science*, pages 291–306. Springer.

Anli, A. (2006). *Méthodologie de développement des systèmes d'information personnalisés : Application à un système d'information au service des usagers des transports terrestres de personnes*. Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambresis.

Annett, J. et Duncan, K. (1967). Task analysis and training design. *Occupational Psychology*, 41:211–227.

Antoniou, G. et van Harmelen, F. (2004). Web Ontology Language : OWL. *Handbook on Ontologies*, pages 67–92.

Arabshian, K. et Schulzrinne, H. (2006). Distributed context-aware agent architecture for global service discovery. *In Proceedings of the Second International Workshop on Semantic Web Technology For Ubiquitous and Mobile Applications (SWUMA'06)*.

ATLAS-INRIA and LINA (2006). *ATL User Manual*. version 0.7.

Azevedo, P., Merrick, R. et Roberts, D. (2000). Ovid to auiml user-oriented interface modelling. *In* Nunes, N., éditeur : *Proceedings of the First International Workshop "Towards a UML Profile for Interactive Systems Development" (TUPIS'00)*, New York.

Bacha, F., Abed, M. et Oliveira, K. (2013). *Progressions and Innovations in Model-Driven Software Engineering*, chapitre Context-Aware MDA Approach for Content Personalization in User Interface Development. IGI Global.

Bacha, F., Oliveira, K. et Abed, M. (2011a). A model driven architecture approach for user interface generation focused on content personalization. *In Proceedings of the Fifth IEEE International Conference on Research Challenges in Information Science, RCIS 2011 (Gosier, Guadeloupe, France, 19-21 May, 2011)*.

Bacha, F., Oliveira, K. et Abed, M. (2011b). Providing personalized information in transport systems : A model driven architecture approach. *In MSLT 2011, First IEEE International Conference on Mobility, Security and Logistics in Transport*, pages 452–459.

Bacha, F., Oliveira, K. et Abed, M. (2011c). Supporting models for the generation of personalized user interfaces with uiml. *In Software Support for User Interface Description Language*, UIDL 2011 (Interact 2011 workshop). Lisbon, Portugal.

Bacha, F., Oliveira, K. et Abed, M. (2011d). Transformation des modèles de bpmn vers uiml. *In Actes des 7èmes Journées sur l'Ingénierie Dirigée par les Modèles, IDM 2011*, pages 77–83, Lille-France.

Bacha, F., Oliveira, K. et Abed, M. (2011e). Using contfext modeling and domain ontology in the design of personalized user interface. *International Journal on Computer Science and Information Systems (IJCSIS)*, 6:69–94.

Balbo, S., Ozkan, N. et Paris, C. (2004). Choosing the right task modelling notation : A taxonomy. *In The Handbook of Task Analysis for Human-Computer Interaction*, pages 445–465. Lawrence Erlbaum Associates.

Baldauf, M., Dustdar, S. et Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

Barthet, M. F. et Tarby, J. C. (1996). The diane+ method. *In* Vanderdonckt, J., éditeur : *Computer-aided design of user interfaces*, pages 95–120, Namur, Belgium. Presses Universitaires de Namur.

Becker, C. et Dürr, F. (2005). On location models for ubiquitous computing. *Personal Ubiquitous Comput.*, 9(1):20–31.

Berners-Lee, T., Hendler, J. et Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

Berti, S., Correani, F., Mori, G., Paterno, F. et Santoro, C. (2004). Teresa : a transformation-based environment for designing and developing multi-device interfaces. *In CHI Extended Abstracts'04*, pages 793–794.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A. et Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161 – 180.

Bodart, F., Provot, I., Leheureux, J.-M., Hennebert, A., Vanderdonckt, J. et Zucchinetti, G. (1996). Key activities for a development methodology of interactive applications. *In Chapter 7 in "Critical Issues in User Interface Systems Engineering" , D. Benyon and Ph. Palanque (Eds.)*, pages 109–134. Springer-Verlag.

Bolchini, C., Curino, C. A., Quintarelli, E., Schreiber, F. A. et Tanca, L. (2007). A data-oriented survey of context models. *SIGMOD Rec.*, 36(4):19–26.

Börzsönyi, S., Kossmann, D. et Stocker, K. (2001). The skyline operator. *In Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, Washington, DC, USA. IEEE Computer Society.

Bouchelligua, W., Mahfoudhi, A., Benammar, L., Rebai, S. et Abed, M. (2010). An mde approach for user interface adaptation to the context of use. *In HCSE*, volume 6409 de *Lecture Notes in Computer Science*, pages 62–78. Springer.

BPMI (2004). *Business Process Model and Notation Version 1.0.*

Bradley, N. (1998). *The XML Companion.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st édition.

Bradley, N. A. et Dunlop, M. D. (2005). Toward a multidisciplinary model of context to support context-aware computing. *Hum.-Comput. Interact.*, 20(4):403–446.

Brickley, D. et Guha, R. V. (2004). Rdf vocabulary description language 1.0 : Rdf schema. Rapport technique.

Brossard, A. (2008). *PERCOMOM : une méthode de modélisation des applications interactives personnalisées appliquée à l'information voyageur dans le domaine des transports collectifs.* Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambresis.

Brossard, A., Abed, M. et Kolski, C. (2007). Modélisation conceptuelle des ihm. une approche globale s'appuyant sur les processus métier. *Ingénierie des Systèmes d'Information*, 12(5):69–108.

Brossard, A., Abed, M. et Kolski, C. (2011). Taking context into account in conceptual models using a model driven engineering approach. *Information & Software Technology*, 53(12):1349–1369.

Brown, P., Bovey, J. et Chen, X. (1997). Context-aware applications : from the laboratory to the marketplace. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 4(5):58–64.

Brown, P. J. (1996). The Stick-e Document : a Framework for Creating Context-aware Applications. *In Proceedings of EP'96, Palo Alto*, pages 259–272. also published in it EP–odd.

Burke, R. (2002). Hybrid recommender systems : Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

Bézivin, J. (2005). On the unification power of models. *Software and System Modeling*, 4(2):171–188.

Bézivin, J. et Gerbé, O. (2001). Towards a precise definition of the omg/mda framework. *In ASE*, pages 273–280. IEEE Computer Society.

Calvary, G., Coutaz, J. et Thevenin, D. (2000). Embedding plasticity in the development process of interactive systems. *In 6th ERCIM Workshop "User Interface for All", Florence, CNR Firenze*.

Calvary, G., Coutaz, J. et Thevenin, D. (2001). A unifying reference framework for the development of plastic user interfaces. *In IFIP WG2.7 (13.2) Working Conference, EHCI01,Toronto, Springer Verlag Publ., LNCS 2254, M. Reed Little, L. Nigay Eds*, pages 173–192.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. et Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L. et Vanderdonckt, J. (2002). Plasticity of user interfaces : A revised reference framework. *In First International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2002, Bucarest*, pages 127–134.

Calvary, G., de Wasseige, O., Faure, D. et Vanderdonckt, J. (2011). User interface extensible markup language sig. *In* Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P. et Winckler, M., éditeurs : *Human-Computer Interaction, INTERACT 2011*, volume 6949 de *Lecture Notes in Computer Science*, pages 693–695. Springer Berlin / Heidelberg.

Calvary, G., Demeure, A., Coutaz, J. et Dâassi, O. (2004). Adaptation des interfaces homme-machine à leur contexte d'usage plasticité des ihm. *Revue d'Intelligence Artificielle*, 18(4):577–606.

Card, S. K., Moran, T. P. et Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):396–410.

Card, S. K., Newell, A. et Moran, T. P. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates Inc., Hillsdale, NJ, USA.

Chen, H., Perich, F., Finin, T. W. et Joshi, A. (2004). Soupa : Standard ontology for ubiquitous and pervasive applications. *In MobiQuitous*, pages 258–267. IEEE Computer Society.

Chomicki, J. (2002). Querying with intrinsic preferences. *In Proceedings of the 8th International Conference on Extending Database Technology : Advances in Database Technology*, EDBT '02, pages 34–51, London, UK, UK. Springer-Verlag.

Christopher R., E., Marc, A., Brendan D., B. et Stuart C., M. (2011). Developing usability engineering tool suites : Towards identifying productivity gains. *In Proceedings of Human Systems Integration Symposium 2011*, Vienna. American Society of Naval Engineers (ASNE).

Cingil, I., Dogac, A. et Azgin, A. (2000). A broader approach to personalization. *Communication of the ACM*, pages 136–141.

Clerckx, T., Luyten, K. et Coninx, K. (2005). Dynamo-aid : A design process and a runtime architecture for dynamic model-based user interface development. *In* Bastide, R., Palanque, P. et Roth, J., éditeurs : *Engineering Human Computer Interaction and Interactive Systems*, volume 3425 de *Lecture Notes in Computer Science*, pages 871–876. Springer Berlin / Heidelberg.

Coninx, K., Luyten, K., Vandervelpen, C., den Bergh, J. V. et Creemers, B. (2003). Dygimes : Dynamically generating interfaces for mobile computing devices and embedded systems. *In* Chittaro, L., éditeur : *Mobile HCI*, volume 2795 de *Lecture Notes in Computer Science*, pages 256–270. Springer.

Cuppens, E., Raymaekers, C. et Coninx, K. (2005). A model-based design process for interactive virtual environments. *In* Gilroy, S. W. et Harrison, M. D., éditeurs : *DSV-IS*, volume 3941 de *Lecture Notes in Computer Science*, pages 225–236. Springer.

Czarnecki, K. et Helsen, S. (2003). Classification of model transformation approaches. *In Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*.

Czarnecki, K. et Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645.

Dey, A. K. (2000). *Providing architectural support for building context-aware applications*. Thèse de doctorat, Georgia Institute of Technology.

Dey, A. K. et Abowd, G. D. (2000). Towards a better understanding of context and context-awareness. *In Workshop on The What, Who, Where, When, and How of Context-Awareness (CHI 2000)*, The Hague, The Netherlands.

Dey, A. K., Abowd, G. D. et Wood, A. (1999). Cyberdesk : a framework for providing self-integrating context-aware services. *Knowledge-Based Systems*, 11(1):3–13.

Dey, K., Salber, D. et Abowd, G. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction*, 16(2-4):97–166.

Diaw, S., Lbath, R. et Coulette, B. (2010). État de l'art sur le développement logiciel basé sur les transformations de modèles. *Technique et Science Informatiques*, 29(4-5):505–536.

Dieterich, H., Malinowski, U., Kühme, T. et Schneider-Hufschmidt, M. (1993). *State of the art in adaptive user interfaces*. North-Holland, Amsterdam.

Doucet, A., Lumineau, N., Berrut, C., Denos, N., Rumpler, B., Rocacher, D., Boughanem, M., Soule-Dupuy, C., Mouaddib, N. et Kostadinov, D. (2004). Action Spécifique sur la Personnalisation de l'Information. rapport de fin de contrat, Groupe MRIM - CLIPS-IMAG.

Dourish, P. (2004). What we talk about when we talk about context. *Personal Ubiquitous Computing*, 8:19–30.

Dyche, J. (2001). *The Crm Handbook*. Pearson Education, Limited.

Eichholz, C., Dittmar, A. et Forbrig, P. (2005). Using task modelling concepts for achieving adaptive workflows. *In Proceedings of the 2004 international conference on Engineering Human Computer Interaction and Interactive Systems*, EHCI-DSVIS'04, pages 96–111, Berlin, Heidelberg. Springer-Verlag.

Eisenstein, J., Vanderdonckt, J. et Puerta, A. R. (2000). Adapting to mobile contexts with user-interface modeling. *In WMCSA*, pages 83–92. IEEE Computer Society.

Eisenstein, J., Vanderdonckt, J. et Puerta, A. R. (2001). Applying model-based techniques to the development of uis for mobile computers. *In IUI*, pages 69–76.

Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M. et Klein, M. (2000). OIL in a nutshell. *In* Dieng, R. et Corby, O., éditeurs : *Knowledge Engineering and Knowledge Management ; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000*, numéro LNCS 1937 de Lecture Notes in Artificial Intelligence, pages 1–16, Juan-les-Pins, France. Springer-Verlag.

FIPA (2001). Device ontology specification. http ://www.fipa.org/specs/fipa00091/-PC00091A.html.

Flaten, R. (2007). Universal design of electronic forms for mobile phones : Accommodating the cognitively disabled.

Florins, M., Simarro, F. M., Vanderdonckt, J. et Michotte, B. (2006). Splitting rules for graceful degradation of user interfaces. *In Proceedings of the working conference on Advanced visual interfaces*, AVI '06, pages 59–66, New York, NY, USA. ACM.

Frasincar, F. et Houben, G.-J. (2002). Hypermedia presentation adaptation on the semantic web. *In Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, AH '02, pages 133–142, London, UK, UK. Springer-Verlag.

Gajos, K. et Weld, D. S. (2004). Supple : automatically generating user interfaces. *In IUI '04 : Proceedings of the 9th international conference on Intelligent user interface*, pages 93–100, New York, NY, USA. ACM Press.

Gajos, K. Z. (2008). *Automatically Generating Personalized User Interfaces*. Thèse de doctorat, University of Washington, Seattle, WA, USA.

Gajos, K. Z., Weld, D. S. et Wobbrock, J. O. (2010). Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174(12-13):910–950.

García-barrios, V. M., Mödritscher, F. et Gütl, C. (2005). Personalisation versus adaptation ? a user-centred model approach and its application. *In* K., T. et H., M., éditeurs : *Proceedings of the International Conference on Knowledge Management*, pages 120–127.

García, J. G., González-Calleros, J. M., Vanderdonckt, J. et Arteaga, J. M. (2009). A theoretical survey of user interface description languages : Preliminary results. *In LA-WEB/CLIHC*, pages 36–43. IEEE Computer Society.

García, J. G., Lemaigre, C., González-Calleros, J. M. et Vanderdonckt, J. (2008). Model-driven approach to design user interfaces for workflow information systems. *Journal of Universal Computer Science*, 14(19):3160–3173.

Gómez-Pérez, A., Fernández-López, M. et Corcho, O. (2004). *Ontological Engineering*. Springer, Berlin.

Gray, W. D., John, B. E. et Atwood, M. E. (1993). Project Ernestine : Validating a GOMS analysis for predicting and explaining real-world task performance. *Human-Computer Interaction*, 8(3):237–309.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

Guarino, N. (1997). Semantic matching : Formal ontological distinctions for information organization, extraction, and integration. *In* Pazienza, M. T., éditeur : *SCIE*, volume 1299 de *Lecture Notes in Computer Science*, pages 139–170. Springer.

Guarino, N. (1998). Formal ontology and information systems. pages 3–15. IOS Press.

Guarino, N. et Giaretta, P. (1995). Ontologies and knowledge bases : Towards a terminological clarification. *In* Mars, N. J. I., éditeur : *Towards Very Large Knowledge Bases : Knowledge Building and Knowledge Sharing*, pages 25–32. IOS Press, Amsterdam.

Hachani, S., Dupuy-Chessa, S. et Front, A. (2009). Une approche générique pour l'adaptation dynamique des ihm au contexte. *In IHM*, ACM International Conference Proceeding Series, pages 89–96. ACM.

Hackos, J. T. et Redish, J. C. (1998). *User and task analysis for interface design*. John Wiley & Sons, Inc., New York, NY, USA.

Hadzic, M. et Chang, E. (2005). Ontology-based support for human disease study. *In Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 6 - Volume 06*, HICSS '05, pages 143.1–, Washington, DC, USA. IEEE Computer Society.

Hagen, P. (1999). Smart personalization. Rapport technique, Forrester Research, Cambridge.

Han, J., Cho, Y. et Choi, J. (2005). Context-aware workflow language based on web services for ubiquitous computing. *In* Gervasi, O., Gavrilova, M., Kumar, V., Laganà, A., Lee, H., Mun, Y., Taniar, D. et Tan, C., éditeurs : *Computational Science and Its Applications ICCSA 2005*, volume 3481 de *Lecture Notes in Computer Science*, pages 249–259. Springer Berlin / Heidelberg.

Harmonia (2002). Generic uiml vocabulary specification. Rapport technique, Harmonia, Inc.

Harmonia-Group (2012). *LiquidApps® 7.1 - Quick-Start Guide*. version 7.1.

Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A. et Vanderdonckt, J. (2009). Human-Centered engineering of interactive systems with the user interface markup language. *In* Seffah, A., Vanderdonckt, J. et Desmarais, M. C., éditeurs : *Human-Centered Software Engineering*, Human-Computer Interaction Series, chapitre 7, pages 139–171. Springer-Verlag, London, UK.

Hewett, B., Card, C., Gasen, M., Perlman, S. et Verplank (1992). Acm sigchi curricula for human-computer interaction. Rapport technique, Association for Computing Machinery, Inc.

Hobbs, J. R. et Pan, F. (2006). Time ontology in owl. http ://www.w3.org/TR/owl-time/.

Hoekstra, R. (2009). *Ontology Representation - Design Patterns and Ontologies that Make Sense*, volume 197 de *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Ioannidis, Y. et Koutrika, G. (2005). Personalized systems : models and methods from an ir and db perspective. *In Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 1365–1365. VLDB Endowment.

ISO (1989). Iso 8807 :1989 information processing systems – open systems interconnection – lotos – a formal description technique based on the temporal ordering of observational behaviour.

Jameson, A. (2001). *Systems That Adapt to Their Users : An Integrative Perspective*. Thèse de doctorat, Saarbrücken : Saarland University.

Javier, F. (2010). *A Development Method for User Interfaces of Rich Internet Applications*. Thèse de doctorat, Université catholique de Louvain - Belgique.

Jordan, B. (1996). Chapter 3 ethnographic workplace studies and cscw. *In* Dan Shapiro, M. T. et TraunmÃŒller, R., éditeurs : *The Design of Computer Supported Cooperative Work and Groupware Systems*, volume 12 de *Human Factors in Information Technology*, pages 17 – 42. North-Holland.

Jouault, F. et Kurtev, I. (2005). Transforming models with atl. *In Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*, Montego Bay, Jamaica.

Jouault, F. et Kurtev, I. (2006). On the architectural alignment of atl and qvt. *In Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1188–1195, New York, NY, USA. ACM.

Kappel, G., Retschitzegger, W. et Schwinger, W. (2000). Modeling customizable web applications - a requirement's perspective. *In Proceedings of the International Conference on Digital Libraries*, Kyoto, Japan.

Karp, P. D., Chaudhri, V. K. et Thomere, J. (1999). Xol : An xml-based ontology exchange language. Rapport technique, SRI International.

Kieras, D. (1997). A Guide to GOMS Model Usability Evaluation using NGOMSL. *In* Helander, P. P. M., éditeur : *Handbook of Human-Computer Interaction*, chapitre 31. Elsevier Science B.V.

Kiessling, W. (2002). Foundations of preferences in database systems. *In Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 311–322. VLDB Endowment.

Kim, E. et Choi, J. (2006). An ontology-based context model in a smart home. *In* Gavrilova, M. L., Gervasi, O., Kumar, V., Tan, C. J. K., Taniar, D., Laganà, A., Mun, Y. et Choo, H., éditeurs : *ICCSA (4)*, volume 3983 de *Lecture Notes in Computer Science*, pages 11–20. Springer.

Kim, W. (2002). Personalization : Definition, status, and challenges ahead. *Journal of Object Technology*, 1(1):29–40.

Kleppe, A. G., Warmer, J. et Bast, W. (2003). *MDA Explained : The Model Driven Architecture : Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Kobsa, A., Koenemann, J. et Pohl, W. (2001). Personalized hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review*, 16(2):111–155.

Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H. et Malm, E.-J. (2003). Managing context information in mobile devices. *IEEE Pervasive Computing*, 2(3):42–51.

Kostadinov, D. (2008). *Personnalisation de l'information : une approche de gestion de profils et de reformulation de requêtes*. Thèse de doctorat, Université de Versailles Saint-Quentin-en-Yvelines.

Koutrika, G. et Ioannidis, Y. E. (2005). Constrained optimalities in query personalization. *In* Özcan, F., éditeur : *SIGMOD Conference*, pages 73–84. ACM.

Kristiansen, R. et Trætteberg, H. (2007). Model-based user interface design in the context of workflow models. *In* Winckler, M., Johnson, H. et Palanque, P., éditeurs : *Task Models and Diagrams for User Interface Design*, volume 4849 de *Lecture Notes in Computer Science*, pages 227–239. Springer, Berlin / Heidelberg.

Lacroix, M. et Lavency, P. (1987). Preferences ; putting more knowledge into queries. *In Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 217–225, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Lassila, O. et McGuinness, D. L. (2001). The role of frame-based representation on the semantic web. Rapport technique KSL-01-02, Stanford University, Stanford.

Ledoux, T. (2001). État de l'art sur l'adaptabilité. Rapport technique, Research Report RNTL ARCAD.

Li, Y., Hong, J. I. et Landay, J. A. (2007). Design challenges and principles for wizard of oz testing of location-enhanced applications. *IEEE Pervasive Computing*, 6(2):70–75.

Lieberman, H. et Selker, T. (2000). Out of context : Computer systems that adapt to, and learn from, context. *IBM Systems Journal*, 39:617–632.

Limbourg, Q. et Vanderdonckt, J. (2003). *Comparing Task Models for User Interface Design*. Lawrence Erlbaum Associates, Mahwah.

Limbourg, Q. et Vanderdonckt, J. (2004). Usixml : A user interface description language supporting multiple levels of independence. *In* Matera, M. et Comai, S., éditeurs : *ICWE Workshops*, pages 325–338. Rinton Press.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. et Lopez-Jaquero, V. (2005). Usixml : A language supporting multi-path development of user interfaces. *Engineering Human Computer Interaction and Interactive Systems*, 3425:200–220.

Lin, X., Li, S., Xu, J., Shi, W. et Gao, Q. (2005). An efficient context modeling and reasoning system in pervasive environment : Using absolute and relative context filtering technology. *In* Fan, W., Wu, Z. et Yang, J., éditeurs : *Advances in Web-Age Information Management*, volume 3739 de *Lecture Notes in Computer Science*, pages 357–367. Springer.

Louhichi, S., Graiet, M., Kmimech, M., Bhiri, M. T., Gaaloul, W. et Cariou, E. (2011). Atl transformation for the generation of sca model. *In Proceedings of the 2011 Seventh International Conference on Semantics, Knowledge and Grids*, SKG '11, pages 164–167, Washington, DC, USA. IEEE Computer Society.

Luke, S. et Heflin, J. (2000). Shoe 1.01 proposed specification. Rapport technique, SHOE Project.

Luyten, K., Clerckx, T., Coninx, K. et Vanderdonckt, J. (2003). Derivation of a dialog model from a task model by activity chain extraction. *In* Jorge, J. A., Nunes, N. J. et e Cunha, J. F., éditeurs : *DSV-IS*, volume 2844 de *Lecture Notes in Computer Science*, pages 203–217. Springer.

Luyten, K. et Coninx, K. (2004). Uiml.net : an open uiml renderer for the .net framework. *In* Jacob, R. J. K., Limbourg, Q. et Vanderdonckt, J., éditeurs : *CADUI*, pages 257–268. Kluwer.

Luyten, K., Thys, K., Vermeulen, J. et Coninx, K. (2006). A generic approach for multi-device user interface rendering with uiml. *In* Calvary, G., Pribeanu, C., Santucci, G. et Vanderdonckt, J., éditeurs : *CADUI*, pages 175–182. Springer.

Manning, C. D., Raghavan, P. et Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

Marcal de Oliveira, K., Bacha, F., Mnasser, H. et Mourad, A. (2013). Transportation ontology definition and application for the content personalization of user interfaces. *Expert Systems with Applications*, 40(8):3145–3159.

Mccarthy, J. (1993). Notes on formalizing context. *In IJCAI*, pages 555–562, San Mateo, California. Morgan Kaumann.

McGuinness, D. L. et Van Harmelen, F. (2004). OWL Web Ontology Language overview. *W3C Recommendation*, 10:1–19.

Meixner, G., Breiner, K. et Seissler, M. (2011). *Model-Driven Useware Engineering*, pages 1–26. Studies in Computational Intelligence, SCI. Springer, Heidelberg.

Mellor, S. J., Clark, A. N. et Futagami, T. (2003). Guest editors' introduction : Model-driven development. *IEEE Software*, 20:14–18.

Mens, T. et Gorp, P. V. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142.

Merrick, R., Wood, B. et Krebs, W. (2004). Abstract user interface markup language. *In* Luyten, K., Abrams, M., Vanderdonckt, J. et Limbourg, Q., éditeurs : *Developing User Interfaces with XML Advances on User Interface Description Languages*, pages 39–46.

Meskens, J., Luyten, K. et Coninx, K. (2010). Jelly : a multi-device design environment for managing consistency across devices. *In* Santucci, G., éditeur : *AVI*, pages 289–296. ACM Press.

Meskens, J., Vermeulen, J., Luyten, K. et Coninx, K. (2008). Gummy for multi-platform user interface designs : shape me, multiply me, fix me, use me. *In* Levialdi, S., éditeur : *AVI*, pages 233–240. ACM Press.

Miller, B., Konstan, J. et Riedl, J. (2004). PocketLens : Toward a Personal Recommender System. *ACM Transactions on Information Systems*, 22(3):437–476.

Minsky, M. L. (1969). *Semantic Information Processing*. The MIT Press.

Mnasser, H., Khemaja, M., Oliveira, K. et Abed, M. (2010). A public transportation ontology to support user travel planning. *In* Loucopoulos, P. et Cavarero, J.-L., éditeurs : *RCIS*, pages 127–136. IEEE.

Mobasher, B., Cooley, R. et Srivastava, J. (2000). Automatic personnalization based on web usage mining. *Communication of the ACM*, 8(43):142–151.

Mori, G., Paterno, F. et Santoro, C. (2002). Ctte : Support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28(9):1–17.

Mori, G., Paterno, F. et Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30:507–520.

Myers, B., Hudson, S. E. et Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. et Swartout, W. (1991). Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56.

Noy, N. et Rector, A. (2006). Defining N-ary Relations on the Semantic Web. Rapport technique, World Wide Web Consortium.

Noy, N. F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R. W. et Musen, M. A. (2001). Creating semantic web contents with protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71.

OMG (2003). *MDA Guide Version 1.0.1.*

OMG (2006). *Meta Object Facility (MOF) Core Specification Version 2.0.*

OMG (2007). *XML Metadata Interchange (XMI).* OMG.

OMG (2012). *OMG Object Constraint Language (OCL)- Version 2.3.1.* Object Management Group.

Oztürk, P. et Aamodt, A. (1997). Towards a model of context for case-based diagnostic problem solving. *In Proceedings of the Interdisciplinary Conference on Modeling and Using Context*, pages 198–208.

Pascoe, J. (1998). Adding generic contextual capabilities to wearable computers. *In ISWC*, pages 92–99.

Paterno, F. (1999). *Model-Based Design and Evaluation of Interactive Applications.* Springer-Verlag, London, UK, UK, 1st édition.

Paterno, F. (2000). *Model-based design and evaluation of interactive applications.* Springer, London.

Paterno, F. (2002). Task models in interactive software systems. *Handbook of Software Engineering and Knowledge Engineering*, 1:1–19.

Pérez, E., Fortier, A., Rossi, G. et Gordillo, S. (2009). Rethinking context models. *In Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems : ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK 2009*, OTM '09, pages 78–87, Berlin, Heidelberg. Springer-Verlag.

Pessoa, R. M., Calvi, C. Z., Filho, J. G. P., de Farias, C. R. G. et Neisse, R. (2007). Semantic context reasoning using ontology based models. *In Proceedings of the 13th open European summer school and IFIP TC6.6 conference on Dependable and adaptable networks and services*, EUNICE'07, pages 44–51, Berlin, Heidelberg. Springer-Verlag.

Phanouriou, C. (2000). *Uiml : a device-independent user interface markup language.* Thèse de doctorat. AAI9991276.

Pontico, F., Farenc, C. et Winckler, M. (2006). Model-based support for specifying eservice egovernment applications. *In* Coninx, K., Luyten, K. et Schneider, K. A., éditeurs : *TAMODIA*, volume 4385 de *Lecture Notes in Computer Science*, pages 54–67. Springer.

Preuveneers, D., den Bergh, J. V., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V. et Bosschere, K. D. (2004). Towards an extensible context ontology for ambient intelligence. *In* Markopoulos, P., Eggen, B., Aarts, E. et Crowley, J. L., éditeurs : *Second European Symposium on Ambient Intelligence*, volume 3295 de *LNCS*, pages 148 – 159, Eindhoven, The Netherlands. Springer.

Rosenberg, M. (2001). The personalization story.

Rossi, G., Schwabe, D. et Guimarães, R. (2001). Designing personalized web applications. *In Proceedings of 10th International World Wide Web Conference*, pages 275–284, Hong Kong. ACM Press.

Rousseau, B., Browne, P., Malone, P. et ÒFoghlù, M. (2004). User profiling for content personalisation in information retrieval. *In 19th ACM Symposium on Applied Computing, SAC'04*, Nicosie, Chypre.

Rukzio, E., Noda, C., Luca, A. D., Hamard, J. et Coskun, F. (2008). Automatic form filling on mobile devices. *Pervasive and Mobile Computing*, 4(2):161–181.

Ryan, N., Pascoe, J. et Morse, D. (1997). Enhanced reality fieldwork : the context-aware archaeological assistant. *In* Gaffney, V., van Leusen, M. et Exxon, S., éditeurs : *Computer Applications and Quantitative Methods in Archaeology (CAA 97), Oxford*.

Samaan, K. (2006). *Prise en Compte du Modèle d'Interaction dans le Processus de Construction et d'Adaptation d'Applications Interactives*. These, Ecole Centrale de Lyon.

Schilit, B., Adams, N. et Want, R. (1994). Context-aware computing applications. *In Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, WMCSA '94, pages 85–90, Washington, DC, USA. IEEE Computer Society.

Schmidt, A., Beigl, M. et Gellersen, H.-W. (1999). There is more to context than location. *Computers and Graphics*, 23(6):893–901.

Seidewitz, E. (2003). What Models Mean. *IEEE Software*, 20(5):26–32.

Sendin, M., Lopez-Jaquero, V. et Collazos, C. A. (2008). Collaborative explicit plasticity framework : a conceptual scheme for the generation of plastic and group-aware user interfaces. *J. UCS*, 14(9):1447–1462.

Simonin, J. et Carbonell, N. (2007). Interfaces adaptatives adaptation dynamique à l'utilisateur courant. *Computing Research Repository*, abs/0708.3742.

Smith, M. J. et O'Neill, E. J. (1996). Beyond task analysis : exploiting task models in application implementation. *In Conference companion on Human factors in computing systems : common ground*, CHI '96, pages 263–264, New York, NY, USA. ACM.

Sottet, J.-S. (2008). *Méga-IHM : malléabilité des Interfaces Homme-Machine dirigées par les modèles*. Thèse de doctorat. Thèse de doctorat Informatique préparée au Laboratoire d'Informatique de Grenoble (LIG), Université Joseph Fourier.

Sottet, J.-S., Ganneau, V., Calvary, G., Coutaz, J., Favre, J.-M. et Demumieux, R. (2007). Model-driven adaptation for plastic user interfaces. *In Proc. INTERACT 2007, the eleventh IFIP TC13 International Conference on Human-Computer Interaction*, pages 397–410. Springer LNCS (Lecture Notes in Computer Science), Brasil, September 10-14, 2007.

Souchon, N. et Vanderdonckt, J. (2003). A review of xml-compliant user interface description languages. *In DSV-IS*, volume 2844 de *Lecture Notes in Computer Science*, pages 377–391. Springer.

Steinberg, D., Budinsky, F., Paternostro, M. et Merks, E. (2009). *EMF : Eclipse Modeling Framework*. Addison-Wesley, Boston, MA, 2. édition.

Stephanidis, C., Paramythis, A., Sfyrakis, M., Stergiou, A., Maou, N., Leventis, A., Paparoulis, G. et Karagiandidis, C. (1998). Adaptable and adaptive user interfaces for disabled users in avanti project. *In* Triglia, S., Mullery, A., Campolargo, M., Vanderstraeten, H. et Mampaey, M. E., éditeurs : *Proceedings of the 5th International Conference on Intelligence in Services and Networks, Technology for Ubiquitous Telecom Services*, volume 1430 de *Lecture Notes in Computer Science*, pages 153–166. Springer-Verlag, Germany.

Storrs, G. (1995). The notion of task in human-computer interaction. *In Proceedings of the HCI'95 conference on People and computers X*, HCI '95, pages 357–365, New York, NY, USA. Cambridge University Press.

Strang, T. et Popien, C. L. (2004). A context modeling survey. *In UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, Nottingham.

Sure, Y., Angele, J. et Staab, S. (2002). Ontoedit : Guiding ontology development by methodology and inferencing. *In* Meersman, R. et Tari, Z., éditeurs : *CoopIS/-DOA/ODBASE*, volume 2519 de *Lecture Notes in Computer Science*, pages 1205–1222. Springer.

Szekely, P. A., Sukaviriya, P. N., Castells, P., Muthukumarasamy, J. et Salcher, E. (1995). Declarative interface models for user interface construction tools : the mastermind approach. *In Engineering for Human-Computer Interaction, Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, Yellowstone Park, USA, August 1995*, pages 120–150.

Tabary, D. (2001). *Contribution à TOOD, une méthode à base de modèles pour la spécification et la conception des systèmes interactifs*. These, Université de Valenciennes et du Hainaut-Cambresis.

Taconet, C. et Aoul, Z. (2008). Context-awareness and model driven engineering : Illustration by an e-commerce application scenario. *In ICDIM*, pages 864–869. IEEE.

Tesoriero, R. et Vanderdonckt, J. (2010). Extending usixml to support user-aware interfaces. *In* Bernhaupt, R., Forbrig, P., Gulliksen, J. et Larusdottir, M., éditeurs : *HCSE*, volume 6409 de *Lecture Notes in Computer Science*, pages 95–110. Springer.

Thevenin, D. (2001). *L'adaptation en Interction Homme-Machine : le cas de la plasticité*. Thèse de doctorat. Thèse de doctorat Informatique préparée au Laboratoire de Communication Langagière et Interaction Personne-Système (IMAG), Université Joseph Fourier 238 pages.

Thevenin, D. et Coutaz, J. (1999). Plasticity of User Interfaces : Framework and Research Agenda. *Proc Interact99 Edinburgh A Sasse C Johnson Eds IFIP IOS Press Publ*, 99:110–117.

Tolulope, O. K. (2008). Automatic filling of web forms using xml user profiles for context-based web personalization. Rapport technique, University of the West of England.

Touzi, J. (2007). *Aide à la conception de Système d'Information Collaboratif support de l'interopérabilité des entreprises*. Thèse de doctorat, Centre de Génie Industriel - Ecole des Mines d'Albi Carmaux.

Trætteberg, H. (1999). Modelling work : Workflow and task modelling. *In* Vanderdonckt, J. et Puerta, A. R., éditeurs : *Computer-Aided Design of User Interfaces II, Proceedings of the Third International Conference of Computer-Aided Design of User Interfaces, October 21-23, 1999, Louvain-la-Neuve, Belgium*, pages 275–280. Kluwer.

Trætteberg, H. (2002). *Model-based User Interface Design*. Thèse de doctorat, Norwegian University of Science and Technology.

Trewin, S., Zimmermann, G. et Vanderheiden, G. (2004). Abstract representations as a basis for usable user interfaces. *Interacting with Computers*, 16(3):477 – 506.

UMO (2003). User model ontology. http ://www.u2m.org/2003/02/UserModelOntology-.daml.

UsiXML (2007). Usixml (user interface extensible markup language) (version 1.8). Rapport technique, Université catholique de Louvain (Eds) - Belgium.

Vale, S. et Hammoudi, S. (2008). Context-aware model driven development by parameterized transformation. *In MDISIS*.

Van Der Veer, G. C., Lenting, B. F. et Bergevoet, B. A. J. (1996). GTA : Groupware task analysis - modeling complexity. *Acta Psychologica*, 91:297–322.

van Harmelen, F., Patel-Schneider, P. F. et Horrocks, I. (2001). Reference description of the daml+oil (march 2001) ontology markup language. http ://www.daml.org/2001/03/reference.html.

van Heijst, G., Schreiber, A. T. et Wielinga, B. J. (1997). Using explicit ontologies in kbs development. *International Journal Human-Computer Studies*, 46(2):183–292.

Van Setten, M. (2001). *Personalized Information Systems*. Telematica Institut.

Vanderdonckt, J. (2005). A mda-compliant environment for developing user interfaces of information systems. *In* Pastor, O. et e Cunha, J. F., éditeurs : *CAiSE*, volume 3520 de *Lecture Notes in Computer Science*, pages 16–31. Springer.

Vanderdonckt, J., Grolaux, D., Roy, P. V., Limbourg, Q., Macq, B. M. et Michel, B. (2005). A design space for context-sensitive user interfaces. *In* Hurley, R. T. et Feng, W., éditeurs : *IASSE*, pages 207–214. ISCA.

Vanderdonckt, J., Tarby, J.-C. et Derycke, A. (1998). Using data flow diagrams for supporting task models. *In* Markopoulos, P. et Johnson, P., éditeurs : *Design, Specification and Verification of Interactive Systems'98, Supplementary Proceedings of the Fifth International Eurographics Workshop, June 3-5, 1998, Abingdon, United Kingdom*, volume 2, pages 1–16. Eurographics Association.

Vara, J. M. (2009). *M2DAT : a Technical Solution for Model-Driven Development of Web Information Systems*. Thèse de doctorat, ETSII, University Rey Juan Carlos, Madrid, Spain.

W3C (1999). Resource description framework (RDF). model and syntax specification. Rapport technique, W3C.

W3C (2004). Xml and 'the semantic web'. http ://xml.coverpages.org/xmlAndSemanticWeb.html.

W3C (2009a). Delivery context ontology. http ://www.w3.org/TR/2009/WD-dcontology-20090616/.

W3C, W. W. W. C. (2003). Xforms 1.0. http ://www.w3.org/TR/2003/REC-xforms-20031014/.

W3C, W. W. W. C. (2009b). Xforms 1.1. http ://www.w3.org/TR/2009/REC-xforms-20091020/.

Wang, X. H., Zhang, D. Q., Gu, T. et Pung, H. K. (2004). Ontology based context modeling and reasoning using owl. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0:18.

Ward, A., Jones, A. et Hopper, A. (1997). A new location technique for the active office. *IEEE Wireless Communications*, 4:42–47.

Weibenberg, N., Voisard, A. et Gartmann, R. (2004). Using ontologies in personalized mobile applications. *In Proceedings of the 12th annual ACM international workshop on Geographic information systems*, GIS '04, pages 2–11, New York, NY, USA. ACM.

Weyersa, B., Burkolterb, D., Luthera, W. et Klugec, A. (2012). Formal modeling and reconfiguration of user interfaces for reduction of errors in failure handling of complex systems. *International Journal of Human-Computer Interaction*. To appear.

Winograd, T. (2001). Architectures for context. *Human-Computer Interaction*, 16(2): 401–419.

WMCS (1999). *Workflow Management Coalition, Terminology & Glossary (Document Number WFMC-TC-1011)*. Workflow Management Coalition Specification.

Zimmermann, A., Lorenz, A. et Oppermann, R. (2007). An operational definition of context. *In* Kokinov, B. N., Richardson, D. C., Roth-Berghofer, T. et Vieu, L., éditeurs : *Modeling and Using Context, 6th International and Interdisciplinary Conference (CONTEXT 2007)*, volume 4635 de *Lecture Notes in Computer Science*, pages 558–571. Springer.

Zimmermann, G., Vanderheiden, G. et Gilman, A. (2003). Universal remote console - prototyping for the alternate interface access standard. *In Proceedings of the User interfaces for all 7th international conference on Universal access : theoretical perspectives, practice, and experience*, ERCIM'02, pages 524–531, Berlin, Heidelberg. Springer-Verlag.

Une approche MDA pour l'intégration de la personnalisation du contenu dans la conception et la génération des applications interactives

Résumé

Les travaux de recherche présentés dans ce mémoire se situent dans les thématiques de la génération des applications interactives et de la personnalisation du contenu. Cette thèse propose une approche de type MDA (Model Driven Architecture), indépendante du domaine d'application, permettant la conception et la génération semi-automatique des applications interactives à contenus personnalisés, compte tenu des informations sur le contexte d'utilisation et l'ontologie de domaine. Cette approche met en œuvre deux méthodes de personnalisation du contenu, à savoir le remplissage automatique des formulaires et l'enrichissement des requêtes. Pour atteindre cet objectif, nous avons développé la solution technique permettant la conception, la transformation des modèles ainsi que la génération de l'IHM (Interface Homme-Machine) finale.

Mots-clés :

Model Driven Architecture – Personnalisation – Interface Homme-machine – Contexte – Ontologie

---

An MDA approach for content personalization integration in the design and the generation of interactive applications

Abstract

The research work presented in this thesis belongs to the fields of interactive applications generation and content personalization. This thesis proposes an MDA (Model Driven Architecture) approach, independent of the domain application, allowing the design and the semi-automatic generation of personalized content interactive applications. This generation relies on context information and the domain ontology. This approach implements two content personalization methods ; namely the forms auto-filling and the automatic queries enrichment. To achieve this goal, we developed the technical solution allowing the design, the models transformations as well as the generation of the final HCI (Human-Computer Interface).

Keywords :

Model Driven Architecture – Personalization – Human-Computer Interface – Context – Ontology