



HAL
open science

Contribution à une méthode de conception et génération d'interface homme-machine plastique

Mohamad-Anas Hariri

► To cite this version:

Mohamad-Anas Hariri. Contribution à une méthode de conception et génération d'interface homme-machine plastique. Informatique [cs]. Université de Valenciennes et du Hainaut-Crambrésis, 2008. Français. NNT : 2008VALE0018 . tel-03015788

HAL Id: tel-03015788

<https://uphf.hal.science/tel-03015788v1>

Submitted on 10 Dec 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Présentée à

l'Université de Valenciennes et du Hainaut-Cambrésis

en vue de l'obtention du grade de

DOCTEUR

Spécialité Automatique et Informatique des Systèmes Industriels et Humains

Mention Informatique

Par

Mohamad Anas HARIRI

Ingénieur Informatique

***Contribution à une méthode de
conception et génération d'interface
homme-machine plastique***

Soutenance prévue le 30 juin 2008 devant le jury composé de :

G. Calvary	Maître de conférences HDR à l'université Joseph Fourier, Grenoble I	Rapporteur
C. Kolski	Professeur à l'université de Valenciennes	Directeur
S. Lecomte	Professeur à l'université de Valenciennes	Examineur
A. Seffah	Professeur à l'université de Concordia, Canada	Rapporteur
D. Tabary	Maître de conférences à l'université de Valenciennes	Co-Directeur
J. Vanderdonckt	Professeur à l'université catholique de Louvain	Examineur

A mes parents

A ma famille

A mes amis

...

Remerciements

Le travail ayant fait l'objet de cette thèse a été réalisé dans le groupe de recherche « Raisonnement Automatique et Interaction Homme-Machine » (RAIHM), au Laboratoire d'Automatique, de Mécanique et d'Informatique, industrielles et Humaines (LAMIH) CNRS UMR 8530, à l'université de Valenciennes et du Hainaut-Cambrésis.

Je tiens à exprimer mes remerciements en tout premier lieu à mes directeurs de thèse, le Professeur Christophe KOLSKI et Monsieur Dimitri TABARY pour leur collaboration, leur soutien et leurs encouragements tout au long de ces travaux, ainsi que pour la qualité de leur encadrement dans les métiers de la recherche. Je les remercie également pour leur disponibilité, leur patience, leur compréhension et leur complicité, pour avoir su répondre à mes questions et mes doutes, et qui nous ont permis de travailler dans une ambiance conviviale.

J'exprime ma profonde gratitude à Madame Gaëlle CALVARY, Maître de Conférences HDR à l'Université Joseph Fourier, pour les judicieux conseils qu'elle m'a donnés lors des réunions du groupe de travail « CESAME », et pour avoir accepté de lire, critiquer et valider ces travaux.

Je suis très reconnaissant envers Monsieur Jean VANDERDOCKT, Professeur à l'université catholique de Louvain (Belgique), pour l'intérêt qu'il a porté à mes travaux, pour ses discussions enrichissantes lors de ma première proposition, et pour l'honneur qui m'a fait d'avoir participé à ce jury.

Je remercie vivement Monsieur Ahmed SEFFAH, Professeur à l'université de Concordia (Canada), d'avoir examiné ces travaux et d'en avoir été un des rapporteurs. Son expertise reconnue dans le domaine des IHM donne un relief particulier à son jugement sur mes travaux.

Je tiens également à remercier Monsieur Sylvain LECOMTE, Professeur à l'université de Valenciennes et du Hainaut-Cambrésis, pour avoir lu mes travaux et avoir accepté de présider le jury de thèse.

Je remercie chaleureusement tous les membres et ex-membres du LAMIH et de mon équipe RAIHM pour leur accueil, les échanges et les bons moments que nous avons passés ensemble.

Je voudrais remercier mes amis en France, au Canada et en Syrie, qui m'ont toujours soutenu et encouragé.

Je ne pourrai jamais oublier les voix de Jacques Brel et Dalida qui m'accompagnaient pendant mes nuits blanches. Je leur adresse tous mes remerciements, et j'espère qu'ils ont trouvé leur île.

Je tiens à témoigner ma gratitude à Madame Sophie LEPREUX-SIMON, pour ses idées et ses conseils. Elle était toujours présente même aux répétitions malgré le fait d'être dans ses derniers mois de grossesse.

Je remercie ma fille Lucie pour le bonheur qu'elle me procure en la voyant grandir. Je lui adresse toutes mes excuses pour les moments où on l'a laissée à la crèche dès 7h00 du matin pendant ses premiers mois de vie.

Une seule ligne ne saurait transcrire ma gratitude envers mes parents, mes sœurs et mes frères qui m'ont offert leurs encouragements et leur soutien pendant toutes ces années d'études.

Je remercie, enfin, très chaleureusement mon épouse Wafaa pour sa patience, son soutien et ses encouragements pendant ces années et plus particulièrement ces derniers mois. Elle a toujours su m'apaiser durant les moments difficiles. Je lui souhaite de tout cœur de réussir en Amérique du Nord.

Table des matières

REMERCIEMENTS.....	1
TABLE DES MATIERES	3
LISTE DES FIGURES.....	7
LISTE DES TABLEAUX	11
INTRODUCTION GENERALE.....	13
I. CHAPITRE I.....	17
I. INTRODUCTION.....	18
II. MODELES D'ARCHITECTURE DE SYSTEMES INTERACTIFS	18
II.1 Architectures centralisées.....	19
II.2 Architectures réparties.....	20
II.3 Modèles hybrides.....	22
II.4 Conclusion sur les modèles d'architecture	23
III. PATRONS DE CONCEPTION ET COMPOSITION	23
III.1 Patrons de conception.....	23
III.1.1 Catégories de patrons	25
III.1.2 Langages de patrons.....	27
III.1.3 Conclusion sur les patrons de conception	28
III.2 Composants et composition	28
III.2.1 Définition	29
III.2.2 Catégories de composants.....	30
III.2.2.1 Composants métier	31
III.2.2.2 Composants de présentation	32
III.2.3 Types de composition	33
III.2.3.1 Composition statique	34
III.2.3.2 Composition dynamique	35
III.2.4 Approches à base de composants.....	36
III.2.4.1 Catalysis	36
III.2.4.2 Symphony.....	36
III.2.4.3 Select Perspective	38
III.2.4.4 Conclusion sur les approches à base de composants	39
III.2.5 Conclusion sur la composition	39
IV NOUVEAUX SYSTEMES INTERACTIFS ADAPTATIFS ET MOBILES	39
IV.1 Context-Awareness.....	40
IV.2 Exemple informel	40
IV.3 Recherche de nouveaux types d'IHM.....	42
IV.3.1 Plasticité et adaptation	42
IV.3.2 Contexte d'usage.....	44
IV.4 Conclusion sur les nouveaux systèmes interactifs adaptatifs et mobiles.....	44
V. CONCLUSION.....	44
II. CHAPITRE II	47
I. INTRODUCTION.....	48
II. CRITERES D'ETUDE DES METHODES ET DES LANGAGES ETUDIES	48
II.1 Modèle d'interface	48
II.2 Outils.....	50
II.3 Langages supportés.....	50
II.4 Plateformes supportées	50
II.5 Contexte d'usage	51
II.6 Adaptation	51

II.7	<i>Connaissance de conception</i>	51
II.8	<i>Apprentissage</i>	52
II.9	<i>Conclusion sur les critères d'étude</i>	53
III.	ETUDE DE METHODE ET LANGAGES POUR LA SPECIFICATION ET LA GENERATION D'IHM	
	MULTIPLATEFORME ET/OU ORIENTEE CONTEXTE	54
III.1	<i>UIML</i>	55
III.2	<i>AUIML</i>	56
III.3	<i>XIML</i>	58
III.4	<i>XForms</i>	59
III.5	<i>Plastic ML</i>	60
III.6	<i>Projet Rainbow</i>	61
III.7	<i>AMUSING</i>	63
III.8	<i>MOBI-D</i>	64
III.9	<i>Projet Cameleon</i>	66
III.9.1	<i>Teresa</i>	68
III.9.2	<i>UsiXML</i>	69
III.10	<i>L'approche par Comet</i>	73
III.11	<i>Approche de Thevenin</i>	74
III.12	<i>Synthèse</i>	75
IV.	CONCLUSION	80
III.	CHAPITRE III	81
I.	INTRODUCTION	82
II.	NIVEAUX DE L'ADAPTATION	84
III.	METHODE GLOBALE POUR GENERER LES IHM PLASTIQUES	86
III.1	<i>Utilisation d'IDEF0 pour la représentation des activités constitutives de la méthode</i>	86
III.2	<i>Scénario illustratif</i>	89
IV.	NIVEAU ABSTRAIT	90
IV.1	<i>Bibliothèque de vocabulaires</i>	90
IV.2	<i>DTD d'AMXML</i>	92
IV.3	<i>Conversion du MIA en AMXML</i>	93
V	NIVEAU CONCRET	97
V.1	<i>Bibliothèque de composants métier</i>	97
V.2	<i>Patrons de conception</i>	100
V.2.1	<i>Proposition d'un gabarit en vue de l'adaptation</i>	100
V.2.2	<i>Bibliothèques de patrons de conception</i>	103
V.3	<i>Contexte d'usage</i>	104
V.4	<i>Guide de style</i>	105
V.5	<i>Construction de la structure métier</i>	105
V.6	<i>Adaptation primaire</i>	108
VI.	NIVEAU FINAL	112
VI.1	<i>Utilisation du système interactif plastique</i>	112
VI.2	<i>Capture du contexte d'usage</i>	113
VI.3	<i>Processus d'apprentissage</i>	113
VI.4	<i>Adaptation vivante</i>	119
VII.	CONCLUSION	120
IV.	CHAPITRE IV	123
I.	INTRODUCTION	124
II.	PREMIER CAS D'ETUDE : SYSTEME DE GUIDAGE TOURISTIQUE	124
II.1	<i>Présentation de l'application</i>	124
II.2	<i>Modèle de tâche et Modèle d'interface abstraite</i>	125
II.3	<i>Contextes d'usage envisagés</i>	127

II.3.1 Contexte relatif à l'utilisateur	127
II.3.2 Contexte relatif à l'environnement	128
II.3.3 Contexte relatif à la plate-forme	129
II.4 Application de la démarche	131
II.4.1 Construction d'AMXML	131
II.4.2 Construction de la structure métier	135
II.4.3 Adaptation primaire de l'IHM	138
II.4.4 Utilisation du système interactif plastique	142
II.4.5 Adaptation vivante	142
II.4.6 Processus d'apprentissage	142
II.5 Conclusion sur le système de guidage touristique	144
III. SECOND CAS D'ETUDE : SYSTEME DE SUPERVISION D'UNE USINE CHIMIQUE	145
III.1 Présentation de l'application industrielle	145
III.2 Contextes d'usage envisagés	147
III.2.1 Contexte relatif à l'utilisateur	147
III.2.2 Contexte relatif à l'environnement	148
III.2.4 Contexte relatif à la plate-forme	150
III.3 Application de la démarche	151
III.3.1 Construction d'AMXML	151
III.3.2 Construction de la structure métier	158
III.3.3 Adaptation de l'IHM primaire	159
III.3.4 Utilisation du système interactif plastique	162
III.3.5 Adaptation vivante	162
III.3.6 Processus d'apprentissage	163
III.4 Conclusion sur le système de supervision d'une usine chimique	165
IV. CONCLUSION	165
V. CHAPITRE V	167
I. INTRODUCTION	168
II. ORIGINALITE DE LA METHODE PROPOSEE	168
III. POINTS FORTS	169
III.1 Spécification et conception d'IHM plastique	169
III.2 Adaptation	170
III.3 Apprentissage	171
IV. LIMITES	172
V. PERSPECTIVES DE RECHERCHE	173
V.1 À COURT TERME	173
V.1.1 Perspectives relatives à une infrastructure logicielle	174
V.1.2 Perspectives relatives aux patrons et composants	174
V.1.3 Perspectives relatives à l'apprentissage et au capture de contexte	175
V.1.4 Perspectives relatives à la validation de la méthode	175
V.2 À MOYEN TERME	176
V.2.1 Perspectives relatives à l'intégration des nouveaux composants	176
V.2.2 Perspectives relatives à la migration d'IHM	176
V.3 À LONG TERME : RELATIVEMENT A LA PRISE EN COMPTE EXPLICITE DU CONCEPT D'AFFORDANCE	178
VI. CONCLUSION	185
CONCLUSION GENERALE	187
REFERENCES BIBLIOGRAPHIQUES	191
A. ANNEXE A	207
B. ANNEXE B	229
C. ANNEXE C	261
D. ANNEXE D	265

E. ANNEXE E273

Liste des figures

FIGURE I.1 : EVOLUTION DE L'INFORMATIQUE [WEISER, 1991]	18
FIGURE I.2 : MODELE DE SEEHEIM [PFAFF, 1985].....	19
FIGURE I.3 : MODELE ARCH [BASS ET AL., 1991].....	20
FIGURE I.4 : MODELE MVC	21
FIGURE I.5 : MODELE PAC [COUTAZ, 1987].....	21
FIGURE I.6 : ARCHITECTURE LOGICIELLE D'UNE COMET AU SEIN D'UN SYSTEME [CALVARY ET AL., 2004] ...	21
FIGURE I.7 : MODELE PAC-AMODEUS [NIGAY, 1994].....	22
FIGURE I.8 : LES DIFFERENTES COUCHES DU MODELE H ⁴ [GUITTET, 1995].....	23
FIGURE I.9 : ABSTRACTION DE LA SOLUTION PROPOSEE PAR LES PATRONS	25
FIGURE I.10 : DIAGRAMME DE CLASSES UML DE TROIS TYPES DE PATRONS PROPOSES PAR [GAMMA ET AL., 1995] (A) PATRON CONSTRUCTEUR; (B) PATRON FAÇADE ; (C) PATRON VISITEUR....	26
FIGURE I.11 : VUE GENERALE D'UN COMPOSANT D'APRES [HASSINE ET AL., 2002A].....	29
FIGURE I.12 : EXEMPLE D'ARCHITECTURE PROPOSEE PAR [BARAIS ET DUCHIEN, 2004].....	31
FIGURE I.13 : DEUX INTERFACES POUR DEUX MODALITES D'INTERACTION DIFFERENTES	33
FIGURE I.14 : CLASSIFICATION DE LA COMPOSITION DE SYSTEMES EN FONCTION DU TEMPS DE LA COMPOSITION OU LA RECOMPOSITION [MCKINLEY ET AL., 2004A].....	33
FIGURE I.15 : POSSIBILITE DE CHOIX DES CLASSES A L'EXECUTION PAR JVM.....	34
FIGURE I.16 : ARCHITECTURE D'UN COMPOSANT METIER SYMPHONY [HASSINE ET AL., 2002B].....	37
FIGURE I.17 : CYCLE DE VIE EN Y DE LA DEMARCHE SYMPHONY [HASSINE, 2005].....	38
FIGURE I.18 : EXEMPLE DE SYSTEME SENSIBLE AU CONTEXTE	40
FIGURE I.19 : CONTEXTE D'USAGE	41
FIGURE I.20 : ESPACE PROBLEME DE LA PLASTICITE (ADAPTE ET SIMPLIFIE DE [CALVARY ET AL., 2006]).....	43
FIGURE II.1 : ENVIRONNEMENT DE DEVELOPPEMENT D'INTERFACE A BASE DE MODELES (ADAPTE PAR [TABARY, 2001] DE [SZEKELY, 1996]).....	49
FIGURE II.2 : EXEMPLES DE CONNAISSANCES DE CONCEPTION	53
FIGURE II.3 : MODELES D'UIML	55
FIGURE II.4 : EXEMPLE DE COMPARAISON ENTRE UIML ET D'AUTRES LANGAGES.....	56
FIGURE II.5 : ENVIRONNEMENT DE DEVELOPPEMENT AUIML AVEC UN EXEMPLE D'UNE IHM ASSOCIEE A DEUX IHM FINALES DIFFERENTES [WWW.IBM.COM/DEVELOPERWORKS].....	57
FIGURE II.6 : STRUCTURE REPRESENTATIVE DE BASE DE XIIML	58
FIGURE II.7 : STRUCTURE CONCEPTUELLE DE XFORMS	59
FIGURE II.8 : TOOLKIT DE PLASTIC ML	60
FIGURE II.9 : CADRE GENERAL DU PROJET RAINBOW [CHENG ET AL., 2004].....	62
FIGURE II.10 : ARCHITECTURE D'AMUSINGS.....	64
FIGURE II.11 : ARCHITECTURE DE MOBI-D [PUERTA ET EISENSTEIN, 1998].....	66
FIGURE II.12 : APPROCHE A BASE DE MODELES POUR L'ADAPTATION DES INTERFACES (ADAPTEE DE [CALVARY ET AL., 2003] PAR [TARPIN, 2006]).....	67
FIGURE II.13 : VUE GENERALE DE L'ENVIRONNEMENT DE TERESA	68
FIGURE II.14 : MODELES ET OUTILS D'USIXML [STANCIULESCU ET AL., 2005]	70
FIGURE II.15 : EXEMPLE DE TRANSFORMATIONS EN USIXML ENTRE LES NIVEAUX D'ABSTRACTION DE CAMELEON (EXTRAIT DE [VANDERDONCKT ET AL., 2004]).....	70

FIGURE II.16 : TRANSFORMATION DU MODELE DE TACHE VERS LE MODELE DE DOMAINE AVEC L'OUTIL IDEALXML [STANCIULESCU ET AL., 2005]	71
FIGURE II.17 : NIVEAUX DE FIDELITE DE L'IHM [VANDERDONCKT ET COYETTE, 2006]	72
FIGURE II.18 : IHM FINALE GENEREE A PARTIR DES SPECIFICATIONS A NIVEAU DE FIDELITE BASSE EN SKETCHIXML [USIXML.ORG].....	72
FIGURE II.19 : EXEMPLE LIE AU PROCESSUS DE DEVELOPPEMENT D'ARTSTUDIO POUR LA PRODUCTION D'INTERFACES MULTIPLATEFORMES	75
FIGURE III.1 : QUATRE PRINCIPALES ETAPES DE DEVELOPPEMENT DU PROJET CAMELEON	82
FIGURE III.2 : COUVERTURE DE LA PLASTICITE EXPRIMEE SUR L'ESPACE PROBLEME ADAPTE DE LA PLASTICITE (CF. CHAPITRE I, §IV.3.1)	83
FIGURE III.3 : DIAGRAMME D'ACTIVITE EN IDEFO.....	83
FIGURE III.4 : NIVEAU TEMPOREL DE L'ADAPTATION.....	85
FIGURE III.5 : DETECTION DE LA PLATE-FORME PENDANT LE PROCESSUS D'ADAPTATION.....	86
FIGURE III.6 : METHODE GLOBALE DE GENERATION DES IHM PLASTIQUES	88
FIGURE III.7 : REPRESENTATION DU SERVEUR DE CINEMA EN CTT	89
FIGURE III.8 : SCHEMA DE L'AMXML (LA DTD COMPLETE SE TROUVE EN ANNEXE A).....	93
FIGURE III.9 : CONVERTIR LE MIA EN AMXML	94
FIGURE III.10 : AMXML DE L'IHM « RECHERCHE PAR ACTEUR ».....	96
FIGURE III.11 : ARCHITECTURE D'UN SYSTEME A BASE DE REGLES [DURKIN, 1994].....	98
FIGURE III.12 : EXEMPLE DE COMPOSANT METIER (CAS OU LE MOTEUR D'INFERENCE EST INTERNE AU COMPOSANT).....	100
FIGURE III.13 : REPRESENTATION EN XML DU PATRON "REGLEUR DE TAILLE ET DE LOCALISATION"	102
FIGURE III.14 : DEUX COMPOSITIONS METIER POSSIBLES.....	104
FIGURE III.15 : EXEMPLE DE STYLES DE DEUX PLATEFORMES DIFFERENTES	105
FIGURE III.16 : CONSTRUCTION DE LA STRUCTURE METIER	107
FIGURE III.17 : STRUCTURE METIER « RECHERCHE PAR ACTEUR ».....	108
FIGURE III.18 : ADAPTATION PRIMAIRE	109
FIGURE III.19 : DEUX LOCALISATIONS DIFFERENTES POUR DEUX IHM DE DOMAINES VARIES	110
FIGURE III.20 ARCHITECTURE DU SYSTEME INTERACTIF PLASTIQUE.....	111
FIGURE III.21 : EXEMPLE DE PREMIERE VERSION DE L'IHM, AVEC SES COMPOSANTS METIER.....	112
FIGURE III.22 : UTILISATION DU SYSTEME INTERACTIF PLASTIQUE	113
FIGURE III.23 : CAPTURE DU CONTEXTE D'USAGE.....	113
FIGURE III.24 : PROCESSUS D'APPRENTISSAGE.....	114
FIGURE III.25 : REGLE D'ADAPTATION	115
FIGURE III.26 : ARBRE DE DECISION CONSTRUIT AVEC C4.5 POUR LA DECISION DE FONCER LA COULEUR DU TEXTE.....	117
FIGURE III.27 : POSSIBILITE DE MODIFIER L'IHM A L'EXECUTION PAR L'UTILISATEUR	118
FIGURE III.28 : BASE DE CONNAISSANCE AVANT ET APRES LE PROCESSUS D'APPRENTISSAGE	118
FIGURE III.29 : PROCESSUS D'ADAPTATION VIVANTE.....	119
FIGURE III.30 : L'IHM DE LA TACHE « RECHERCHE PAR ACTEUR » READAPTEE SELON LA TAILLE D'ECRAN DU TELEPHONE MOBILE	120
FIGURE IV.31 : GUIDE DE VISITE "GEOVISITE" PROPOSE PAR FRANCE TELECOM	125
FIGURE IV.32 : MODELE DE TACHE DU SYSTEME DE GUIDAGE TOURISTIQUE REPRESENTE EN CTT	126
FIGURE IV.33 : MODELE D'INTERFACE ABSTRAITE DE LA TACHE RACINE EN USIXM (A L'AIDE DE L'OUTIL IDEALXML CF. CHAPITRE II, § III.9.2).....	127

FIGURE IV.34 : PARTIE DU MODELE DE TACHE CONCERNANT LA SOUS-TACHE « NAVIGUER VERS »	131
FIGURE IV.35 : MODELE D'INTERFACE ABSTRAITE DE LA TACHE « NAVIGUER VERS » EN USIXML	132
FIGURE IV.36 : AMXML DE L'IHM « NAVIGUER VERS »	134
FIGURE IV.37 : APPLICATION DES REGLES DE GENERATION DE LA STRUCTURE METIER	135
FIGURE IV.38 : STRUCTURE METIER « NAVIGUER VERS »	137
FIGURE IV.39 : PREMIERE VERSION DU SYSTEME DE GUIDAGE TOURISTIQUE « NAVIGUER VERS »	140
FIGURE IV.40 : PARTIE DE L'IHM DU SYSTEME DE GUIDAGE TOURISTIQUE ADAPTEE A LA CATEGORIE D'AGE DU VISITEUR.....	141
FIGURE IV.41 : CHANGEMENT DE L'ORIENTATION D'AFFICHAGE DE PDA	142
FIGURE IV.42 : CHANGEMENT DE LOCALISATION DES BOUTONS QUIT ET OK	143
FIGURE IV.82 : UNE PARTIE DE L'ARBRE DE DECISION DU PATRON ARRANGEUR, AVANT LE PROCESSUS D'APPRENTISSAGE.....	143
FIGURE IV.44 : UNE PARTIE DE L'ARBRE DE DECISION DU PATRON ARRANGEUR, APRES LE PROCESSUS D'APPRENTISSAGE.....	144
FIGURE IV.84 : FAÇONNAGE ET REMPLISSAGE DE GODETS	146
FIGURE IV.85 : ORDRE PARTIEL DE TYPES D'ECHELLES [PETERSEN ET MAY, 2003]	150
FIGURE IV.86 : MODELE DE TACHE DYNAMIQUE DE LA TACHE « RECEVOIR ALERTE » EN TOOD	152
FIGURE IV.87 : FICHE DESCRIPTIVE DE LA TACHE « RECEVOIR ALERTE »	153
FIGURE IV.88 : DEUX ZONES D'INTERACTION POUR LES DEUX SOUS-TACHES T1.1 ET T1.2	154
FIGURE IV.89 : TRADUCTION EN CTT DU MODELE DE TACHE "RECEVOIR ALERTE" TOOD.....	156
FIGURE IV.90 : MODELE D'INTERFACE ABSTRAITE DE LA TACHE "RECEVOIR ALERTE" EN USIXML.....	157
FIGURE IV.91 : AMXML DE L'IHM « RECEVOIR ALERTE »	157
FIGURE IV.92 : APPLICATION DES REGLES DE GENERATION DE LA STRUCTURE METIER	158
FIGURE IV.93 : STRUCTURE METIER « RECEVOIR ALERTE »	159
FIGURE IV.94 : ARBRE DE DECISION DU COMPOSANT "DONNEECTRL" POUR DECIDER DES ECHELLES DE MESURE NECESSAIRES.....	160
FIGURE IV.95 : PREMIERE VERSION DU SYSTEME DE SUPERVISION INDUSTRIELLE « RECEVOIR ALERTE ».....	161
FIGURE IV.96 : IHM ADAPTEE POUR UN PDA, REGENERATION POUR UN TELEPHONE PORTABLE	163
FIGURE IV.97 : IHM MODIFIEE PAR L'UTILISATEUR	164
FIGURE IV.98 : ARBRE DE DECISION DU COMPOSANT "DONNEECTRL" AMELIORE	164
FIGURE V.1 : INTERPRETEUR DE PATRON DE CONCEPTION COGENT [BUDINSKY ET AL. 1996].....	173
FIGURE V.2 : EXEMPLE DE COMPOSANT METIER COMPOSITE.....	175
FIGURE V.3 : EXEMPLE DE COMPOSANT WCOMP (CAPTEUR ET CONTROLE DE L'ALLUMAGE) [CHEUNG ET AL., 2003]	176
FIGURE V.4 : PROPOSITION D'UNE MANIERE DE MIGRER L'IHM (COTE UTILISATEUR).....	177
FIGURE V.5 : METHODE GLOBALE DE GENERATION D'IHM PLASTIQUE	180
FIGURE V.6 : CONVERTIR LE MIA EN AMXML	180
FIGURE V.7 : CONSTRUCTION DE LA STRUCTURE METIER	181
FIGURE V.8 : DEUX COMPOSITIONS METIER POSSIBLES.....	181
FIGURE V.9 : ADAPTATION PRIMAIRE	182
FIGURE V.10 : UTILISATION DU SYSTEME INTERACTIF PLASTIQUE	183
FIGURE V.11 : CAPTURE DU CONTEXTE D'USAGE.....	183
FIGURE V.12 : PROCESSUS D'APPRENTISSAGE	184
FIGURE V.13 : PROCESSUS D'ADAPTATION VIVANTE.....	184
FIGURE AC.1 : PADEV : OUTIL DE DEVELOPPEMENT DE PATRONS DE CONCEPTION.....	262
FIGURE AC.2 : FACILITE DE DESCRIPTION D'UNE SOLUTION EN PROPOSANT AU DEVELOPPEUR UNE LISTE DE BALISES.....	263
FIGURE AC.3 : POSSIBILITE D'IMPORTER UNE SOLUTION DEVELOPEE DANS UN LANGAGE DE PROGRAMMATION	264

FIGURE AE.1 : EXEMPLE D'APPLICATION DE L'ADAPTATION VIVANTE.....274

Liste des tableaux

TABLEAU II.1 : METHODES ET LANGAGES POUR LA SPECIFICATION ET LA GENERATION D'IHM MULTIPLATEFORME ET/OU ORIENTEE CONTEXTE –PARTIE I	78
TABLEAU II.2 : METHODES ET LANGAGES POUR LA SPECIFICATION ET LA GENERATION D'IHM MULTIPLATEFORME ET/OU ORIENTEE CONTEXTE- PARTIE II	79
TABLEAU III.1 : SYNTHESE ENTRE AMXML ET DES MIA D'AUTRES METHODES.....	91
TABLEAU III.2 : GABARIT PERMETTANT DE DEFINIR LES PATRONS ASSOCIES A LA METHODE (LA DTD COMPLETE SE TROUVE EN ANNEXE B)	101
TABLEAU IV.3 : FACETTES DU CONTEXTE RELATIF A L'UTILISATEUR	128
TABLEAU IV.4 : FACETTES DU CONTEXTE RELATIF A L'ENVIRONNEMENT	129
TABLEAU IV.5 : FACETTES DU CONTEXTE RELATIF A LA PLATE-FORME	131
TABLEAU IV.6 : FACETTES DU CONTEXTE RELATIF A L'UTILISATEUR	148
TABLEAU IV.7 : FACETTES DU CONTEXTE RELATIF A L'ENVIRONNEMENT	149
TABLEAU IV.8 : ÉCHELLES DE MESURE [STEVENS, 1946].....	149
TABLEAU IV.9 : VALEURS EVENTUELLES DE LA TEMPERATURE T_{CURF} SELON DIFFERENTES ECHELLES DE MESURE (ADAPTE DE [PETERSEN ET MAY, 2006])	150
TABLEAU IV.10 : FACETTES DU CONTEXTE RELATIF A LA PLATE-FORME	151

Introduction générale

De nouvelles perspectives d'utilisation des systèmes interactifs se sont ouvertes suite aux progrès technologiques et à l'apparition de nouveaux terminaux de travail mobiles, et avec l'évolution à tous les niveaux des moyens d'information et de communication. Informatique pervasive, ubiquitaire ou encore diffuse est le terme pour décrire le troisième âge de l'informatique [Schmidt, 1999]. Cette percée des nouvelles technologies de l'information et de la communication a contribué au lancement de recherches sur une nouvelle génération de systèmes interactifs : l'IHM doit être adaptée à son contexte d'usage, tout en faisant en sorte que cette opération n'engendre pas des coûts de développement et maintenance importants lors de chaque adaptation. Celle-ci consiste à prendre en considération des informations relatives à la plate-forme, à l'utilisateur et à l'environnement d'utilisation, en préservant l'utilisabilité de l'IHM.

La plasticité d'une IHM [Thevenin et Coutaz, 1999 ; Calvary *et al.*, 2001a ; 2001b ; 2003] dénote la capacité de l'IHM à s'adapter aux variations du contexte d'usage dans le respect de l'utilisabilité. La norme ISO 9241-11 (qui fait partie de la série ISO 9241) définit l'utilisabilité de la manière suivante : « un système est utilisable lorsqu'il permet à l'utilisateur de réaliser sa tâche avec efficacité, efficience et satisfaction dans le contexte d'utilisation spécifié ». Cette définition comporte plusieurs dimensions :

- *Efficacité et efficience* : le système doit permettre à l'utilisateur de réaliser toutes les actions nécessaires à l'atteinte de son but, ceci de façon rapide.
- *Satisfaction* : les logiciels utilisés sont-ils agréables à utiliser, l'utilisateur doit-il faire un effort supplémentaire important pour utiliser le système ?
- *Contexte d'usage* : le système doit être compatible avec différents contextes d'usage.

Les cas élémentaires de plasticité sont ceux où l'adaptation ne s'effectue que sur un seul type de contexte (en particulier le contexte lié à la plate-forme). En dehors de ces cas élémentaires, la plasticité introduit des cas plus complexes d'adaptation à différents contextes, de manière statique ou dynamique. Dans ce domaine de recherche, des travaux de recherche essaient de proposer des méthodes et des approches orientées contexte en se basant sur des mécanismes et des outils existants et aussi sur des nouvelles technologies. La plupart des méthodes de conception orientée contexte et des méthodes supportant les notions de multiplateforme se focalisent sur l'adaptation pré-calculée. Cette dernière peut être supportée par le retour à la conception. Dans ce cas l'IHM est reconstruite de façon adéquate au nouveau contexte en revenant vers le modèle d'interface abstraite voire le modèle de tâche (en cas d'approche à base de modèles). De plus, les méthodes actuelles sont généralement basées sur l'adaptation prédéfinie par le concepteur, sans mécanisme d'amélioration de la qualité de réponse aux changements de contexte.

Dans ce courant de recherche et en partant du concept d'IHM plastique, le travail présenté dans ce mémoire est une contribution à une méthode de conception et génération d'IHM

plastique à partir d'un modèle d'IHM abstraite et/ou un modèle de tâche. Dans notre méthode, l'IHM possède une capacité d'adaptation dynamique, et prend en considération la plus grande gamme possible de chaque élément du contexte d'usage (par rapport à des méthodes et langages orientés contexte). De plus nous proposons qu'un processus d'apprentissage travaille en arrière plan du système ; son but est de préserver l'utilisabilité de l'IHM, en prenant en compte les nouveaux cas du contexte qui ne sont pas considérés pendant la génération de la base de connaissance de système.

Ce mémoire, organisé en cinq chapitres, débute par un chapitre intitulé "Des architectures des systèmes interactifs aux nouveaux systèmes adaptatifs et mobiles". Ce chapitre présente sans souci d'exhaustivité, mais plutôt de représentativité, les principes des modèles d'architecture les plus cités et étudiés actuellement. Nous présentons dans ce premier chapitre les trois principales catégories d'architecture dédiées à la conception des IHM : les architectures centralisées, les architectures réparties et les architecture hybrides. Ensuite, nous présentons les patrons de conception, qui seront utilisés comme une solution aidant à l'adaptation de l'IHM. Nous procédons à un tour d'horizon des approches orientées composants en montrant les divers types de composants et de compositions architecturales. Nous terminerons ce chapitre en présentant les systèmes adaptatifs et mobiles, la plasticité avec son espace problème.

Le deuxième chapitre, appelé "Méthodes et langages pour la spécification et la génération d'IHM multiplateforme et/ou orientée contexte", s'intéresse aux méthodes et langages de spécification et de génération d'IHM possédant différentes capacités d'intégration de la notion de multiplateforme et de multi-contexte. Les méthodes et langages se différencient par leur capacité à établir une IHM capable de s'adapter automatiquement ou semi-automatiquement, par le type d'adaptation adopté, par les mécanismes d'adaptation ou encore par les outils qui supportent chaque approche. Nous décrivons des critères sur lesquelles une comparaison est établie entre les méthodes et langages.

Dans ces deux premiers chapitres, notre travail consiste en une présentation de l'état de l'art sur, d'une part les architectures des systèmes interactifs et de nouveaux types d'IHM, et d'autre part sur les méthodes et langages de spécification et génération d'IHM orientées multiplateforme et/ou orientées contexte. Cet état de l'art nous offre un cadre d'étude pour la proposition d'une méthode de conception et génération d'IHM plastique, présentée au troisième chapitre.

En effet, intitulé "Proposition d'une méthode de conception d'IHM plastique, basée sur des patrons de conception", le troisième chapitre présente notre principale contribution : la méthode proposée s'appuie sur la notion de patrons de conception. Ceux-ci sont utilisés au niveau du passage à l'IHM concrète et pendant l'adaptation. L'architecture du système s'appuie sur une composition basée sur les composants métier. Ceux-ci ont la capacité de changer dynamiquement leur facette de présentation. Cela est adopté comme une des solutions de l'adaptation dynamique au contexte d'usage. Notre méthode s'appuie également sur la notion d'apprentissage. L'intégration d'une technique d'apprentissage

permet de continuer à développer la base de connaissance du système à l'exécution afin de préserver l'utilité de l'adaptation.

Le quatrième chapitre est intitulé "Simulation d'implémentation de la démarche sur deux cas d'étude". Dans celui-ci, nous validons et implémentons la démarche proposée en l'appliquant sur deux études de cas de système interactif plastique, destinés à deux environnements d'utilisation différents. Le premier est un système de guidage touristique qui permet d'indiquer le chemin à suivre à des visiteurs et leur délivre des informations sur la visite. Le deuxième exemple est un système de supervision d'une usine chimique. Ce système surveille et contrôle les données de conduite de l'installation (température, humidité, alimentation électrique, stocks, fonctionnement des machines...). Ce système prévient les utilisateurs en cas d'alerte ou d'incidents dans un but d'intervention. L'illustration de la méthode sur ces deux cas d'étude différents et complémentaires permettra de distinguer les points forts et les points faibles de la méthode. Chacun des deux exemples a ses propres caractéristiques, ses contextes d'usage spécifiques, et sa fréquence de changement du contexte.

Finalement, le cinquième chapitre appelé "Discussion et perspectives de recherches", présente, d'une part les conclusions relatives à la méthode proposée et aux deux études menées, et d'autre part les perspectives de recherches complémentaires à effectuer par rapport à ce travail dans le domaine de la plasticité des IHM.

Chapitre I

**Des architectures des systèmes interactifs aux
nouveaux systèmes adaptatifs et mobiles**

I. Introduction

Dans le rapport entre l'homme et l'informatique, il y a eu globalement trois époques, selon [Weiser, 1991]. Au début des années cinquante, plusieurs personnes étaient connectées à un ordinateur. Puis on est passé à la notion d'ordinateur personnel - il était question de l'interaction entre une personne et son ordinateur. Il s'agit maintenant d'exprimer que l'informatique se retrouve et se retrouvera de plus en plus partout (Figure I.1).

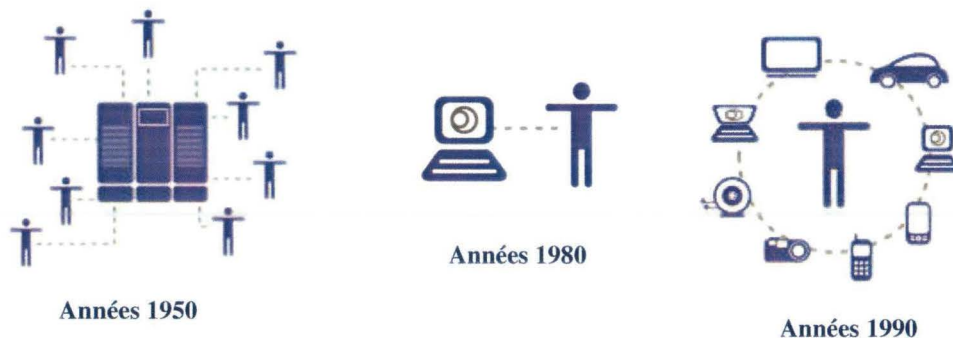


Figure I.1 : Evolution de l'informatique [Weiser, 1991]

La conception des IHM sensibles aux changements du contexte d'usage (P : plate-forme, U : utilisateur, et E : environnement), fait l'objet de nombreuses études de recherche actuellement [Calvary *et al.*, 2002b, 2003, 2005] [Limbourg *et al.*, 2004] [Mori *et al.*, 2003] [Thevenin et Coutaz, 1999].

Nos travaux s'intéressent à la spécification et à la génération des IHM adaptables au contexte d'usage (ce qui sera détaillé dans la section IV). L'adaptation peut se dérouler pendant la conception du système ou à l'exécution. Nous préciserons, dans ce chapitre, le principe des systèmes interactifs adaptatifs et mobiles en nous concentrant sur leur partie interactive. Tout d'abord, nous détaillerons les différents types d'architectures de systèmes interactifs, notamment les approches centralisées, réparties et hybrides. Ensuite nous présenterons les patrons de conception, qui seront utilisés comme une solution aidant à adapter l'IHM au changement contextuel. Nous procéderons à un tour d'horizon des approches orientées composants en montrant les divers types de composants et de compositions architecturales. Nous terminerons ce chapitre en présentant les systèmes adaptatifs et mobiles du point de vue de la plasticité ainsi que l'espace problème de cette dernière.

II. Modèles d'architecture de systèmes interactifs

Les modèles généraux d'architecture de systèmes interactifs cherchent à apporter des solutions aux difficultés de réalisation et de maintenance de leur partie interactive. Ils apparaissent dans un certain nombre d'outils logiciels dont le but est d'apporter une aide dans le maquettage et/ou la réalisation d'interface homme-machine. Tout modèle d'architecture satisfait à plusieurs critères importants [Nigay, 1994] : modifiabilité,

réduction de la complexité, décomposition du travail, intégration des outils de mise en œuvre. Chaque modèle vise à permettre d'identifier les éléments significatifs qui entrent dans la composition des systèmes interactifs, ainsi que les relations qui les lient.

Nous présenterons dans cette section les principes des modèles d'architecture des systèmes interactifs les plus cités et étudiés actuellement. Cette présentation nous permettra par la suite de construire un cadre d'étude pour l'introduction de notre architecture dédiée aux systèmes interactifs **plastiques** [Calvary *et al.*, 2001a] présentée dans le chapitre 3.

II.1 Architectures centralisées

Les modèles centralisés tentent de répondre à une question primordiale : comment isoler la partie interface du reste de l'application, afin de rendre cette dernière indépendante de tout matériel ou de tout outil spécifique à la présentation. Le principal avantage de la centralisation est la possibilité de gérer de manière satisfaisante des aspects tels que le contrôle d'accès ou le verrouillage de données répliquées [Le Pape et Gañçarski, 2004].

Le modèle Langage [Foley *et al.*, 1984] est le premier modèle d'architecture de système interactif. Ce modèle est basé sur l'analogie entre l'interaction homme-machine et le dialogue humain. Comme précisé dans [Coutaz, 1990], Foley et Van Dam étaient les premiers à plaquer une vue linguistique sur l'interaction homme-ordinateur. Il assure la communication de l'application avec l'utilisateur. Le modèle Langage distingue quatre niveaux d'abstraction de l'interface : le niveau conceptuel, le niveau sémantique, le niveau syntaxique, et le niveau lexical. Le modèle Langage présente quatre inconvénients majeurs [Coutaz, 1990] : la priorité est donnée à la forme de l'information, le traitement des formes est centralisé, le langage d'interaction est découpé en deux sous-ensembles distincts, et le niveau d'abstraction des protocoles de communication entre les composants est trop imprécis.

Puis, le modèle *Seeheim* a été la première approche qui a exprimé le modèle Langage sous une forme utilisable. Le modèle de *Seeheim* [Pfaff, 1985] raffine la structuration de l'interface en trois composantes : **La présentation** est la couche en contact direct avec les entrées et sorties. Elle interprète les actions de l'utilisateur (dispositifs physiques) et génère les sorties (affichage) au niveau lexical. **Le contrôleur de dialogue** est l'intermédiaire entre l'utilisateur et le système interactif. **L'interface avec l'application** sert de passerelle entre le noyau fonctionnel et le contrôleur de dialogue. Il ajuste les différences entre les formalismes utilisés par ces deux composants.

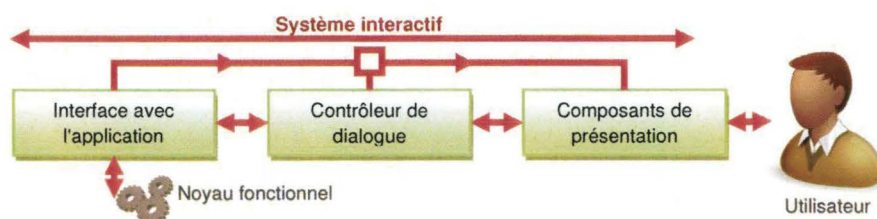


Figure I.2 : Modèle de Seeheim [Pfaff, 1985]

Ensuite, le modèle *ARCH* [Bass *et al.*, 1991], extension du modèle de *Seeheim*, a été proposé. Il est néanmoins plus pragmatique, car il s'inspire des architectures concrètes des systèmes interactifs, avec les notions de plate-forme et de boîte à outils. Il éclate le composant de présentation et contrôle pour tenir compte des considérations d'implantation en se basant sur une boîte à outils. Il comprend cinq composants (Figure I.3) : composant noyau fonctionnel, composant adaptateur au domaine, composant contrôleur de dialogue, composant de présentation, composant d'interaction.

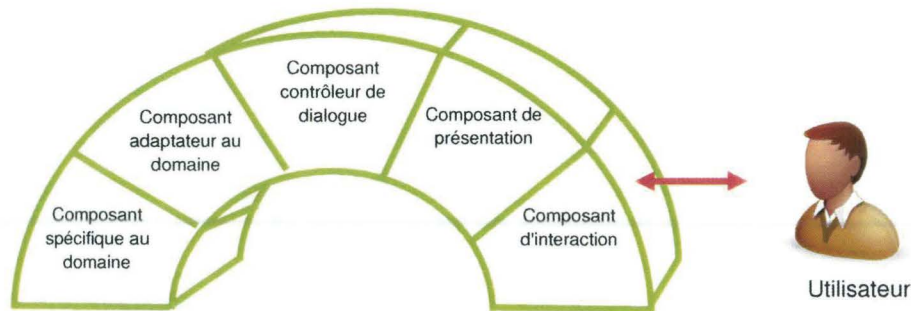


Figure I.3 : Modèle ARCH [Bass *et al.*, 1991]

II.2 Architectures réparties

De telles architectures sont dites aussi à base d'agents. Il s'agit donc de définir la notion d'agent : un agent [Ferber, 1995 ; Bradshaw, 1997] est une entité physique ou virtuelle qui possède des récepteurs et des émetteurs pour acquérir et produire des événements ; selon [Coutaz, 2001] un agent est un vecteur communiquant : il assume un rôle et se manifeste par un comportement observable via l'acquisition et la production d'informations.

Un modèle multi-agents structure un système interactif en un ensemble d'agents qui réagissent à des événements et qui produisent des événements [Coutaz, 1990]. Dans cette section, nous décrivons les principaux modèles à base d'agents *MVC*, *PAC* et *AMF*.

Tout d'abord, le premier modèle à agents est très lié à son environnement de développement et d'exécution [Goldberg, 1984]. Il s'agit du modèle *MVC* (Model, View, Controller) [Schmucker, 1986 ; Krasner et Pope, 1988]. L'idée principale est d'extraire les interactions entre les objets et de réutiliser leurs comportements d'une application à une autre. Selon ce modèle, l'agent est organisé en trois objets (Figure I.4) : **Modèle** qui définit les fonctionnalités propres au domaine et représente le noyau fonctionnel de l'agent ; **Vue** qui maintient une représentation du modèle perceptible par l'utilisateur, qu'elle met à jour à chaque changement d'état du modèle ; **Contrôleur** : il reçoit et interprète les événements utilisateur en les répercutant sur le *Modèle* ou sur la *vue* (retour instantané).

Des recherches portent sur l'amélioration de ce modèle afin de résoudre cet inconvénient. Par exemple, le modèle *MVC2* [Girard et Ribette, 2001] qui regroupe les multiples servlets en une seule (un seul contrôleur) sous l'étiquette *Model 2*.

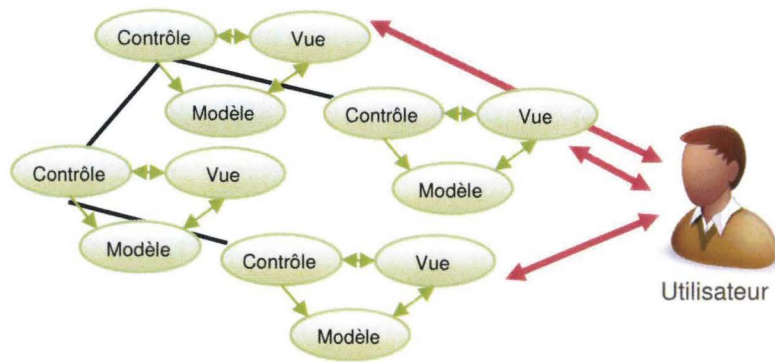


Figure I.4 : Modèle MVC

Ensuite, le deuxième modèle réparti est *PAC* (Présentation, Abstraction, Contrôle). Il a été mis au point par [Coutaz, 1987]. Il permet de décrire le système interactif de manière récursive sous forme d'une hiérarchie d'agents. Chaque entité est décomposée en trois modules, appelés facettes. Les trois facettes (Figure I.5) d'un agent *PAC* sont : **la Présentation**, qui permet de visualiser l'état interne de l'agent, en définissant le comportement perceptible de l'agent auprès de l'utilisateur ; **l'Abstraction** correspondant aux compétences sémantiques de l'agent avec les fonctions et les attributs internes, indépendamment des considérations de présentation ; **le Contrôle** a un double rôle : (1) il joue le rôle de connecteur entre les facettes Présentation et Abstraction de l'agent, (2) il gère des relations avec d'autres agents *PAC*.

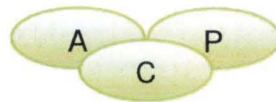


Figure I.5 : Modèle PAC [Coutaz, 1987]

Il est à noter que plusieurs variantes du modèle *PAC* ont été proposées comme *Compact* [Calvary et al., 2005], et *Clover* [Laurillau, 2002]. Le modèle le plus intéressant dans le cadre de notre thèse est le modèle *Compact* (COntext Mouldable PAC for Plasticity) [Calvary et al., 2005]. L'originalité de *Compact* est qu'il essaie de tirer profit simultanément des approches à base de modèles et des architectures logicielles. Il propose de regrouper l'ensemble des spécifications dans le même composant (une *Comet*) en le dotant éventuellement de capacités d'adaptation (cf. Figure I.6).

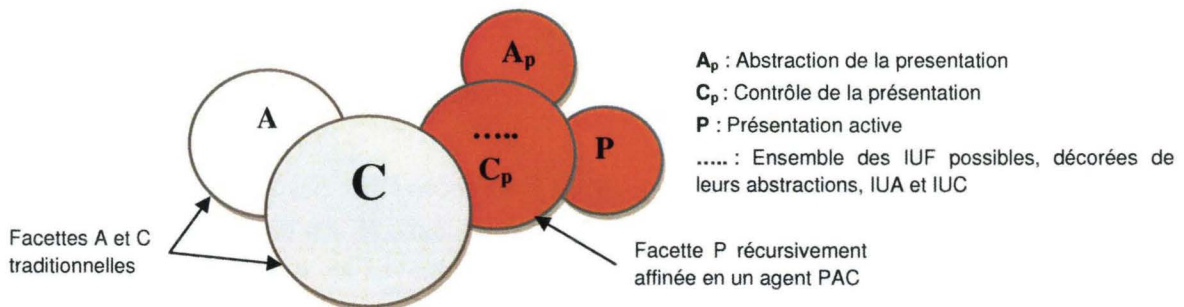


Figure I.6 : Architecture logicielle d'une Comet au sein d'un système [Calvary et al., 2004]

Enfin, le dernier modèle réparti présenté ici est le modèle *AMF* (Agent Multi-Facettes) qui est un modèle d'architecture multi-agents et multi-facettes. Il a été développé à l'école centrale de Lyon par [Ouadou, 1994]. Puis ce modèle a été complété et étendu au travail collaboratif par [Tarpin, 1999]. Le modèle *AMF* est une extension du modèle PAC.

II.3 Modèles hybrides

Les modèles hybrides concilient la décomposition modulaire des modèles centralisés et une description fine du fonctionnement des différents composants. Ils tentent de tirer les avantages de deux types de modèles abordés précédemment. Plusieurs architectures hybrides ont été proposées. Nous présentons ici le modèle *PAC-Amodeus*, le modèle *Multi-couches* et le modèle *H^f*.

Tout d'abord le modèle *PAC-Amodeus* [Nigay, 1994] préconise la décomposition fonctionnelle de *ARCH* et intègre les capacités d'affinement et de prise en compte du parallélisme de *PAC* (Figure I.7). Dans *PAC-Amodeus*, l'application reprend ainsi le découpage en cinq composants décrit dans *ARCH*. Cependant la différence fondamentale vient du fait que le Contrôleur de dialogue est lui-même décomposé en une hiérarchie d'agents *PAC*.

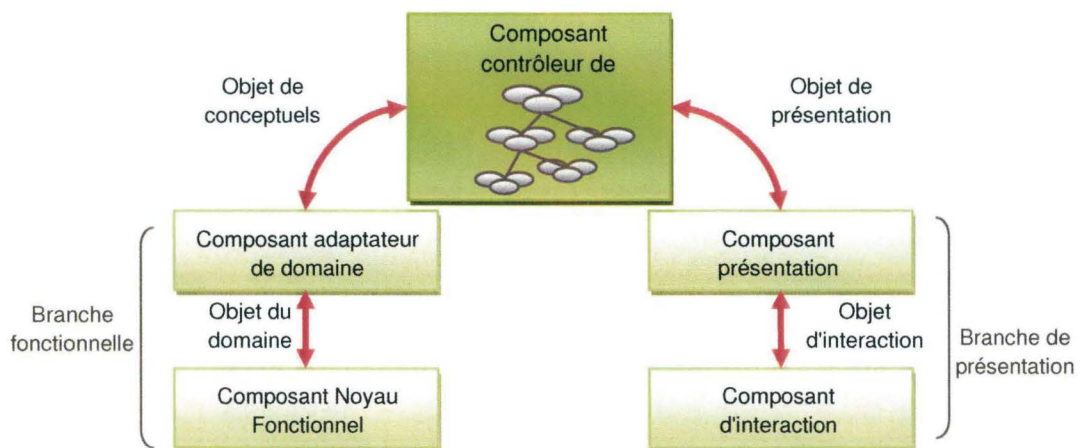


Figure I.7 : Modèle *PAC-Amodeus* [Nigay, 1994]

Ensuite le modèle *Multi-couches* [Fekete, 1996] permet de décomposer l'interface en différentes couches qui peuvent gérer les événements, gérer les objets graphiques ou améliorer les performances. En effet cette décomposition permet d'une part de répartir le dialogue sur plusieurs niveaux. Et d'autre part, cette architecture permet un raffinement des événements.

Puis le Modèle *H^f* [Guittet et Pierra, 1993] a été motivé par des besoins relativement aux systèmes de CAO (Conception Assistée par Ordinateur). *H^f* est basé sur le modèle *ARCH*. Le modèle *H^f* partage de nombreux points communs avec les modèles hybrides présentés précédemment. Son architecture générale ressemble à celle de *PAC-Amodeus*, avec une organisation différente des agents de contrôle, ainsi que du mécanisme de passage de paramètres.

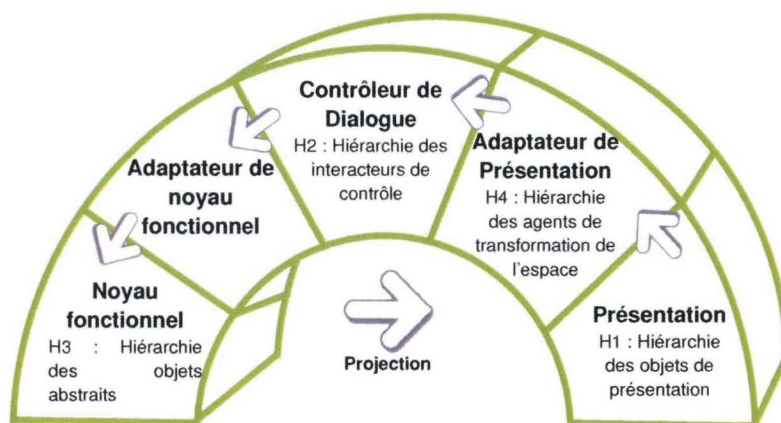


Figure I.8 : les différentes couches du modèle H^4 [Guittet, 1995]

II.4 Conclusion sur les modèles d'architecture

Dans cette partie, nous avons étudié un certain nombre de modèles d'architecture dont l'objectif est de fournir des stratégies de décomposition fonctionnelle et structurelle pour les IHM. Ces modèles centralisés, répartis ou hybrides tentent de faire en sorte que la conception d'une application interactive soit moins coûteuse en temps, en effort et en maintenance. Pour cela, ils préconisent le plus souvent de séparer le Noyau Fonctionnel, la Présentation et le Contrôleur de Dialogue. Alors que les modèles centralisés s'appuient sur une description macroscopique de l'application, en décomposant celle-ci en différents composants, les modèles répartis s'inspirent de l'approche objet pour éclater les éléments de Présentation et de Contrôle de Dialogue à travers toute l'application. Les modèles répartis représentent une organisation d'agents qui travaillent en parallèle et qui coopèrent. Cette approche est particulièrement adaptée aux styles de programmation par objets. L'approche des modèles hybrides tend, quant à elle, à prendre en compte les avantages des approches centralisées et réparties.

Nous présenterons dans la partie suivante, les patrons de conception. Ensuite nous aborderons la composition et les composants d'IHM.

III. Patrons de conception et composition

Dans cette section, nous présenterons la technique des patrons de conception, leurs catégories et leurs langages de spécification. Ensuite nous aborderons les différentes techniques de composition d'IHM, les composants d'IHM et des approches à base de composants.

III.1 Patrons de conception

Un patron de conception (design pattern en anglais) est une solution efficace et éprouvée à un problème de conception et reflète les meilleures pratiques de conception. Comme outil de conception, les patrons s'avèrent plus performants que les guides de conception [Gaffar *et al.*, 2004].

La notion de patron est née d'un architecte [Alexander *et al.*, 1977] pour qui « chaque patron décrit un problème qui se manifeste constamment dans notre environnement et décrit l'architecture de la solution à ce problème, d'une façon telle que l'on puisse réutiliser cette solution des millions de fois sans jamais l'adapter deux fois de la même manière. ». Les patterns ont été utilisés et adaptés à différents domaines d'application dont le génie logiciel et la conception de systèmes d'information. Gamma [Gamma *et al.*, 1994] a proposé un catalogue de patrons de conception basés sur un gabarit donné pour certains problèmes de conception. Une définition de la notion de patron a été proposée par [Johnson, 1997] « Un patron décrit un problème devant être résolu, une solution, et le contexte dans lequel cette solution est considérée. Il nomme une technique et décrit ses coûts et ses avantages. Il permet à une équipe d'utiliser un vocabulaire commun pour décrire leurs modèles ».

Conte et ses collègues [Conte *et al.*, 2001] ont employé des patrons orientés objet pour intégrer la technique de réutilisation pendant les premières phases du cycle de développement de système. Selon eux, un patron transforme un problème en solution ou en nouveaux problèmes : « un patron X emploie un patron Y, si une partie des problèmes résultant de X peut être résolue par Y ». Ensuite, ces auteurs proposent un formalisme pour la manipulation des patrons [Conte *et al.*, 2002a ; 2002b]. Ce travail est employé dans ceux de Godet-Bar et de ses collègues qui ont proposé certains patrons liés à chaque étape du développement d'IHM multimodal [Godet-Bar *et al.*, 2006]. [Van Welie et Van der Veer, 2003] ont classifié des problèmes de conception d'IHM selon différents types (par exemple, problèmes d'interaction, problèmes de navigation, problèmes relatifs à l'expérience de l'utilisateur...), et ont proposé des solutions pour ces problèmes en s'appuyant sur la notion de patron. Chaque patron décrit un problème se reproduisant, son contexte, les points les plus importants à remarquer dans la situation, et une solution au problème. Borchers [Borchers, 2001] suggère une adaptation des patrons qui pourrait être adoptée dans le domaine de l'IHM. Nanard a adapté certains patrons avec une visée de conception des documents hypermédias [Nanard, 2002]. Dans de telles recherches, on s'interroge sur la manière dont un patron pourrait s'adapter aux besoins dynamiques de l'IHM, afin d'être employé dans un nouveau cas plus spécifié. Les patrons de conception constituent une codification intuitive de solutions éprouvées à des problèmes de conception récurrents. Comme tout artéfact réutilisable, l'utilisation d'un patron de conception passe par trois étapes : (1) reconnaître le patron comme étant une solution potentielle au problème de conception, (2) comprendre le patron, sa structure, et les principes qui le sous-tendent, et (3) appliquer le patron au problème. Chacune de ces étapes nécessite une représentation particulière du patron. Un patron expose ses avantages et ses inconvénients suivant son contexte d'application. Les patrons abordent généralement des problèmes élémentaires de conception (création d'objets, structuration des objets ou comportement entre objets) bien que des patrons composés soient également proposés [Riehle, 1997].

La technique des patrons de conception est applicable et utile (voir chapitre III), dans le cas des systèmes adaptatifs dans un environnement non homogène (diverses plateformes,

différents types d'utilisateurs et diverses situations de travail) et changeant fréquemment, qui ont toujours besoin de solutions concernant l'architecture de l'IHM et l'organisation des composants de présentation adéquatement au contexte d'usage. Les patrons de conception sont destinés à résoudre génériquement les problèmes récurrents en conception des systèmes interactifs. Ils décrivent des solutions standard applicables aux problèmes d'architecture et de conception des systèmes adaptatifs. Ces problèmes peuvent être rencontrés à la conception ou pendant l'exécution suivant un changement contextuel. L'avantage de ces patrons est de diminuer le temps nécessaire au développement d'un système et d'augmenter la qualité du résultat, notamment en appliquant des solutions déjà existantes à des problèmes courants de conception.

III.1.1 Catégories de patrons

Les patrons de conception peuvent être variés du point de vue de l'abstraction de l'IHM. Comme par exemple, les patrons de conception pouvant intervenir à la conception d'un système complexe au niveau plutôt abstrait avec des solutions abstraites. Bien qu'au niveau plus concret, les patrons peuvent trouver des solutions plutôt concrètes concernant un contexte particulier (voir Figure I.9).

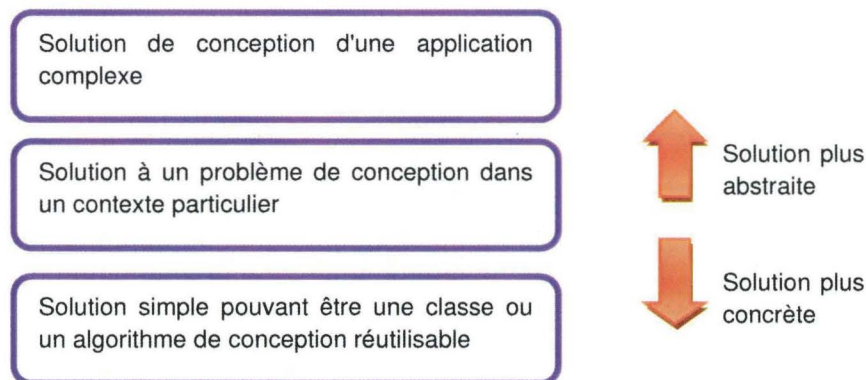


Figure I.9 : Abstraction de la solution proposée par les patrons

Gamma et ses collègues [Gamma *et al.*, 1995] adoptent une approche objet et proposent 23 patrons de conception selon un format très détaillé et spécifique au développement de logiciels. Ces patrons sont classifiés en trois catégories selon leur rôle et leur domaine d'application :

- Les patrons de création qui sont liés au problème de choisir la classe responsable des créations et à l'encapsulation des classes effectives. Par exemple, le patron *Constructeur* permet de séparer la construction d'un objet complexe de sa représentation pour que le même processus de construction puisse créer différentes représentations (cf. Figure I.10, A) ;
- Les patrons structurels qui sont liés aux problèmes d'organisation des objets dans un logiciel. Par exemple, le patron *Façade* peut servir à cacher une conception complexe difficile à comprendre en simplifiant cette complexité et en fournissant

une interface simple du sous-système. Le patron *Façade* définit une interface unifiée à un ensemble d'interfaces de niveau supérieur qui rend le sous-système plus facile à utiliser (cf. Figure I.10, B) ;

- Les patrons comportementaux qui sont liés aux problèmes de communication entre les objets. Par exemple, le patron comportemental *Visiteur* permet d'ajouter une ou plusieurs opérations applicables aux éléments d'une hiérarchie de classes existante, de façon non destructive, en les plaçant dans une autre hiérarchie (cf. Figure I.10, C).

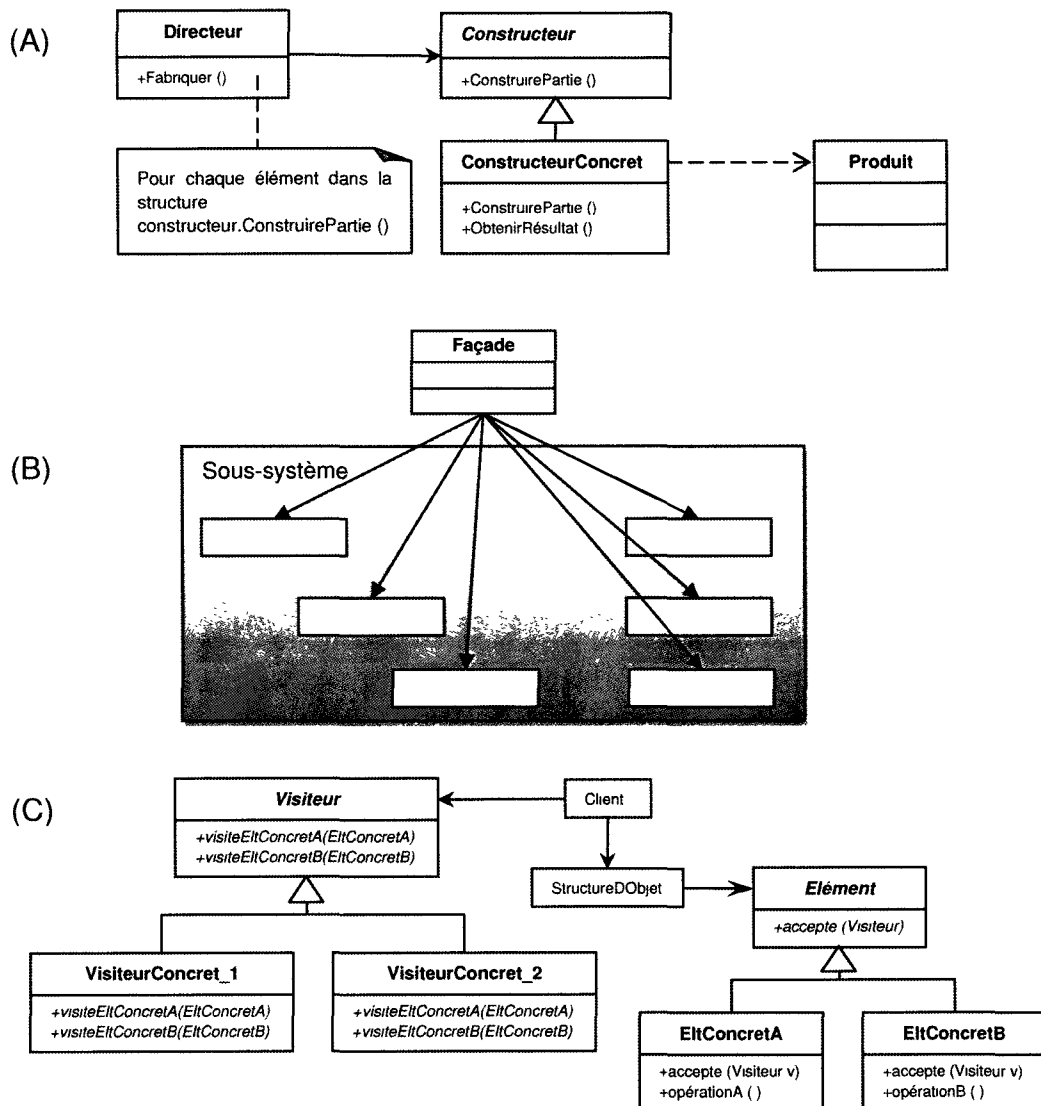


Figure I.10 : Diagramme de classes UML de trois types de patrons proposés par [Gamma et al., 1995] (A) patron Constructeur; (B) patron Façade ; (C) patron Visiteur

Les patrons *GRASP* (*General Responsibility Assignment Software Patterns*) [Larman, 2005] décrivent des principes fondamentaux d'assignation des responsabilités à des objets. Ils sont utilisés comme guide pour assigner correctement les responsabilités aux différentes classes, en liaison avec le modèle *MVC* (cf. Chapitre I, II.2.1). Larman a

proposé une liste de patrons *GRASP* : *Responsabilités*, *Expert*, *Créateur*, *Faible couplage*, etc. ; par exemple le patron *Créateur* permet de déterminer la classe à laquelle la responsabilité de créer les objets est assignée.

[Seffah et Gaffar, 2007] proposent des patrons de conception dédiés au développement à base de modèle. Des patrons supportant différents modèles (modèle de tâche, modèle de dialogue...) ont été proposés. Ces patrons peuvent être aussi personnalisés relativement à un contexte d'utilisation.

Il existe des anti-patrons [Brown *et al.*, 1998] qui sont une forme littéraire décrivant une mauvaise solution à un problème de conception récurrent, dont l'utilisation a des effets négatifs sur la qualité. À l'inverse des patrons de conception, les anti-patrons décrivent ce qu'il ne faut pas faire. Il existe des anti-patrons généraux et ceux spécifiques qui peuvent être classifiés en trois catégories [Moha *et al.*, 2006] : les anti-patrons de développement, d'architecture et de gestion de projet.

Dans le troisième chapitre, nous allons classer les patrons selon leur domaine d'application et la phase à laquelle les patrons peuvent être appliqués.

III.1.2 Langages de patrons

Les patrons facilitent et accélèrent le développement, en fournissant notamment un langage commun aux différents acteurs de ce développement, et favorisent l'évolution, l'adaptation et la réutilisation. Plusieurs auteurs ont cherché à développer des méthodes pour la représentation et la mise en œuvre des patrons.

Gamma et ses collègues [Gamma *et al.*, 1995] ont adopté une représentation uniforme et structurée pour les patrons. Chaque patron est décrit et documenté de la même façon. Les propriétés utilisées sont :

- Nom : chaque patron doit être associé à un nom significatif ;
- Intention : décrit ce que fait le patron, son but, quel problème particulier il résout ;
- Alias : énumère, s'il en existe, d'autres noms communs du patron ;
- Motivation : donne un exemple de problème de conception pouvant être résolu par application du patron. L'illustration par un exemple facilite la compréhension de la description abstraite (donnée par la suite) du patron ;
- Indications d'utilisation : décrit les situations où on peut utiliser le patron ;
- Structure : décrit la structure du patron en utilisant des diagrammes de classes. Des diagrammes de collaboration sont aussi utilisés pour la description de l'interaction entre les classes du patron ;
- Participants : définit les classes (ou objets) intervenant dans le patron et leurs responsabilités ;

- Collaborations : décrit comment les participants interagissent pour assumer leurs responsabilités ;
- Conséquences : décrit comment le patron atteint ses objectifs, quels sont les compromis et les résultats de son utilisation ;
- Implémentation : présente des astuces, techniques et pièges qu'il faut connaître lors de l'implémentation du patron. Cette partie propose aussi, quand il en existe, des solutions typiques du langage utilisé ;
- Exemple de code : donne des fragments de code pour illustrer la façon dont on peut implémenter le patron, par exemple en C++ ou Smalltalk ;
- Utilisations connues : contient des exemples d'utilisation du patron dans des systèmes réels ;
- Patrons apparentés : cite les patrons qui sont reliés avec celui en cours de traitement et leurs différences.

Cette représentation peut être une piste pour proposer un gabarit spécifique pour notre méthode qui sera proposée dans le chapitre III.

III.1.3 Conclusion sur les patrons de conception

Nous avons présenté les catégories de patrons les plus connues, et nous avons présenté une représentation des patrons proposée par Gamma. La notion de patrons de conception peut être adoptée comme une solution adéquate à un environnement d'exécution non-homogène afin de préserver l'utilisabilité de l'IHM. A ce sujet, nous proposerons dans le troisième chapitre un gabarit qui permettra de représenter les patrons de manière à expliciter le problème de conception, le contexte d'utilisation et la solution. Deux catégories des patrons de conception peuvent être envisagées. La première contiendra des patrons pour les problèmes au niveau de la construction de la structure métier (cf. chapitre III, §V.5) du système adaptatif ; la deuxième contiendra des patrons qui préconisent des solutions pour choisir et assembler les composants de présentation pendant l'adaptation de l'IHM, et pour changer les composants de présentation pendant l'adaptation à l'exécution [Savidis, 2005].

III.2 Composants et composition

L'objectif de cette section est de fournir un ensemble d'éléments permettant de comprendre l'approche « Composants » et le développement à base de composants. Le terme « Développement à base de composants » est populaire de nos jours dans le domaine du génie logiciel. L'ingénierie logicielle à base de composants prévoit de construire des applications en assemblant des éléments architecturaux réutilisables. Il suffit de choisir un ensemble de composants qui fournissent certaines fonctionnalités requises par l'application et puis d'assembler ces composants en reliant les entrées aux sorties. L'intégration de composant est le mécanisme permettant de relier des composants entre eux en mettant en relation leurs interfaces respectives. Cependant, contrairement au monde des composants électroniques, l'intégration simple n'est pas suffisante pour assurer

la qualité des interactions entre composants lors de leur déploiement. La composition doit fournir des moyens de détermination des propriétés des assemblages de manière à prévoir leur compatibilité lors du déploiement [Stafford et Wallnau, 2002].

III.2.1 Définition

Plusieurs définitions du concept de composant peuvent être trouvées dans la littérature. Ces définitions diffèrent naturellement en fonction du domaine d'application. Une des définitions la plus souvent énoncée est celle de Szyperski [Szyperski, 1998] pour qui « *un composant est une unité de composition avec des interfaces contractualisées et un contexte de dépendance exprimé explicitement. Un composant peut être déployé indépendamment et il est sujet à composition par une tierce personne* ». Un composant peut être primitif ou composé ; on parle alors dans ce dernier cas de composant composite. La taille de la composition d'un composant peut aller de la fonction mathématique à une application complète. Deux parties définissent un composant. Une première partie, dite externe, comprend la description des interfaces fournies et requises par le composant. Elle définit les interactions du composant avec son environnement. La seconde partie correspond à son contenu et permet la description du fonctionnement interne du composant. La Figure I.11 illustre une vue générale d'un composant selon la définition proposée par WCOP-96 (Workshop on Component Oriented Programming at ECOOP' 96) et citée dans [Hassine *et al.*, 2002a]. Le composant possède une interface de service qui définit les services fournis par le composant et une interface cliente qui représente les collaborations du composant (services acquis).

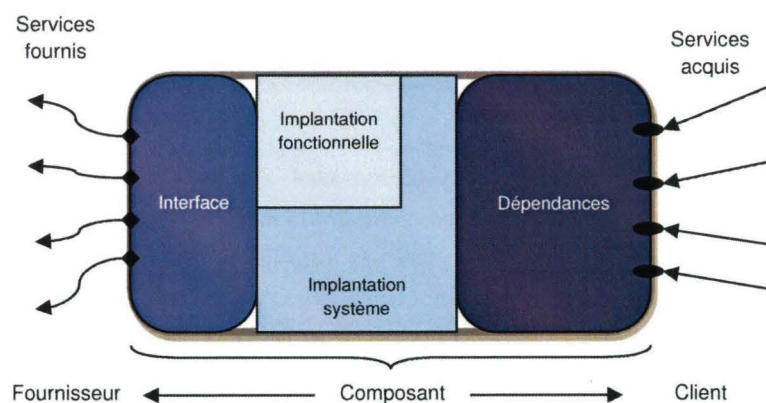


Figure I.11 : Vue générale d'un composant d'après [Hassine *et al.*, 2002a]

L'approche « composant » a été introduite dans l'objectif d'améliorer la réutilisation. Elle consiste à segmenter, rationaliser, encapsuler et plus généralement modulariser les systèmes d'information [Barbier *et al.*, 2002]. Un des intérêts est que les composants sont utilisés plus souvent et deviennent individuellement tous plus fiables. Les développements à base de composants consistent à assembler ces blocs préfabriqués, paramétrables et indépendants pour concevoir un système logiciel. Il n'y a pas vraiment de consensus quant à la définition exacte d'un composant.

Le composant respecte un certain nombre de conventions qui permettent de le manipuler sans connaissances sur son contenu. On pourra ainsi :

- l'importer, le détruire, l'activer ;
- l'assembler avec d'autres composants ;
- le configurer, l'adapter ;
- et le déployer dans un environnement d'exécution.

La réutilisation permet de réduire les coûts et les délais de conception, d'implémentation, de maintenance lorsqu'une traçabilité est associée [Barbier *et al.*, 2002]. La coopération et l'interopérabilité permettraient d'utiliser dans un environnement donné des fonctionnalités complémentaires issues de systèmes différents [Macrelle-Rosselle, 2003]. La valeur ajoutée d'une telle démarche est d'éviter de tout redévelopper soi-même mais plutôt, d'intégrer et d'utiliser des composants existants [Futtersack et Labat, 2000].

III.2.2 Catégories de composants

Au plan conceptuel, nous pouvons différencier les composants de type boîte noire et les composants de type boîte blanche. Les composants de type boîte blanche sont fournis en code source et sont donc modifiables avant intégration. Les composants de type boîte noire, à l'inverse, sont des codes compilés ou binaires ; leurs fonctionnalités ne sont accessibles qu'au moyen de leurs interfaces publiques et ils ne peuvent pas être modifiés directement. Pour étendre ou redéfinir certaines de leurs fonctionnalités, il est nécessaire d'utiliser des adaptateurs. Une solution intermédiaire est un composant dont une partie est ouverte à la modification au moyen d'un ensemble de mécanismes, un tel composant est appelé "*boîte grise*" [Szyperski, 1998].

Nous distinguons généralement deux catégories de composants orientés but : les composants génériques et ceux réutilisables. Dans un composant générique, le but de développement peut être réalisé de plusieurs manières alors que dans un composant réutilisable le but admet un seul scénario. Un scénario décrit une solution dans un contexte prédéfini. Un composant générique ne peut pas être réutilisé directement, il faut préciser le contexte dans lequel il sera réutilisé. A l'inverse, un composant réutilisable fournit une seule solution qui peut directement être réutilisée [Guzelian *et al.*, 2004].

La notion de composant permet de structurer la logique des applications en offrant un support à leur décomposition en unités formant des ensembles cohérents du point de vue fonctionnel. Comme l'illustre la plupart des méthodes d'analyse et de conception, cette décomposition est souvent récursive, ce qui permet de descendre dans le détail sans être submergé par la complexité de l'ensemble du système. Pour permettre une structuration de composants plus forte que celle introduite par les relations d'abonnement aux événements, les composants peuvent être regroupés au sein d'un composant de plus haut niveau : le composant composite. Cette structure récursive permet ainsi de créer des graphes de composants de profondeur arbitraire. Au sein de cette structure, chaque composant nœud

est abonné automatiquement à l'ensemble de ses fils [Hoareau et Mahéo, 2005]. [Barais et Duchien, 2004] propose un exemple d'un composant composite *station* (cf. Figure I.12). Le composant composite *station* est composé d'une *pompe*, d'une *caisse*, et d'un *pistolet*. Chacun de ces trois composants possède des ports pour se relier avec les autres composants (assemblage) afin de former le composant composite *station*. D'autres liens, tels que la délégation, permettent des interactions avec l'extérieur (le client et la banque).

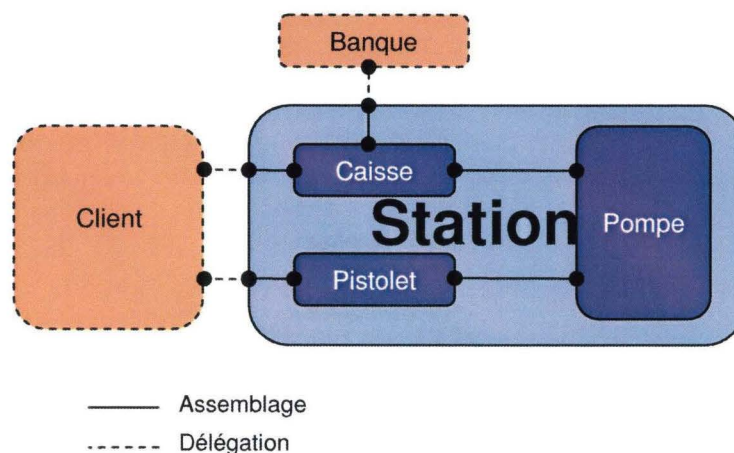


Figure I.12 : Exemple d'architecture proposée par [Barais et Duchien, 2004]

Les composants peuvent être classifiés en fonction de leur niveau d'intégration dans le cycle de développement des systèmes. Nous pouvons distinguer, à ce sujet, les composants métier, et les composants de présentation (IHM). Ces deux catégories de composants seront utilisées dans notre méthode de génération d'IHM plastique (qui sera décrite dans le chapitre III).

III.2.2.1 Composants métier

Aujourd'hui, les recherches dans le domaine de la réutilisation portent sur la conception et l'exploitation de composants conceptuels permettant la réutilisation de connaissances de domaine. Les composants métier expriment des connaissances réutilisables dans le développement de tous les systèmes d'un domaine d'application. Ces composants sont utilisables et réutilisables dès la phase d'analyse du processus de conception car ils expriment souvent une connaissance de haut niveau, orientée sur la définition de problèmes, de besoins. Les composants métier se retrouvent dans cette catégorie car ils couvrent les besoins du domaine [Hassine *et al.*, 2002b].

Une classification à laquelle sont soumis les composants métier est celle des domaines. Wartik et Prieto-Diaz dans [Vestal, 1998] donnent les fondements de la réutilisation de connaissances de domaine. Comme par exemple, dans le domaine de la finance, un compte bancaire avec tout un ensemble d'états et d'opérations type constituant son interface. La spécification une fois des composants pour un domaine, la flexibilité et la généralité des composants pour des usages non forcément prévus, et l'intégration et la réutilisation des composants sont des défis dans l'ingénierie de la réutilisation. De manière

plus générale, l'identification et la classification des domaines correspondent à une discipline à part entière qui agit totalement sur le paradigme même de composant métier. L'OMG¹ (Object Management Group) propose à ce sujet une classification en domaines et sous-domaines. Par exemple, le contrôle de trafic aérien est considéré comme sous-domaine du transport. Dans chaque domaine, l'OMG propose aussi une classification des composants métier en trois types :

- Les composants « processus métier » représentent les activités de gestion du domaine,
- Les composants « entité métier » représentent les concepts du domaine sur lesquels les processus métier sont mis en œuvre,
- Et les composants « utilitaires » sont utilisés par les deux autres types de composants.

III.2.2.2 Composants de présentation

Le découpage des applications en composants logiciels assemblés remet en question la construction de l'interface globale de l'application. L'IHM du système interactif est composée des composants de présentation. Ces derniers font le lien entre le noyau fonctionnel du système et l'utilisateur.

Ils sont chargés ou déchargés sur le dispositif de l'IHM à la conception ou à l'exécution (sous réserve de distribuer avec l'IHM les paquets de composants). Un composant de présentation a un espace d'interaction adapté aux tâches associées au composant. Il contient des informations concrètes et/ou abstraites concernant la modalité d'interaction avec l'utilisateur. Cet espace est spécifié et caractérisé en fonction de la modalité en particulier en fonction de la plateforme cible. Par exemple, dans le cas d'une modalité graphique, un composant de présentation avec une liste de choix peut afficher à l'utilisateur, dans son espace d'interaction, une liste contenant un ensemble de choix, l'utilisateur pouvant alors en choisir un. Alors qu'avec une modalité vocale, le composant de présentation donne à chaque élément de choix un numéro puis il attend l'entrée de l'utilisateur pour détecter son choix (cf. Figure I.13). Les composants de présentation peuvent être réorganisés sur le canevas de l'IHM, cachés ou affichés ; de plus les composants peuvent être reconfigurés (changement de taille, de police, de couleur, etc.) afin de s'adapter au contexte d'usage.

Le composant de présentation a deux comportements à l'exécution : un comportement interne et un autre externe :

- Le comportement interne représente une réaction à un événement. Comme par exemple, quand l'utilisateur appuie sur le bouton de souris et quand le pointeur de

¹ www.omg.org/

souris est sur le composant de présentation, un événement sera déclenché, en collaboration avec des composants fonctionnels du noyau fonctionnel du système. Généralement, le comportement interne ne peut pas être modifié aussi facilement que la présentation du composant.

- Le comportement externe représente la réaction du système au comportement de l'utilisateur. Le comportement externe du composant doit être explicitement décrit par le concepteur avec des rappels de service.

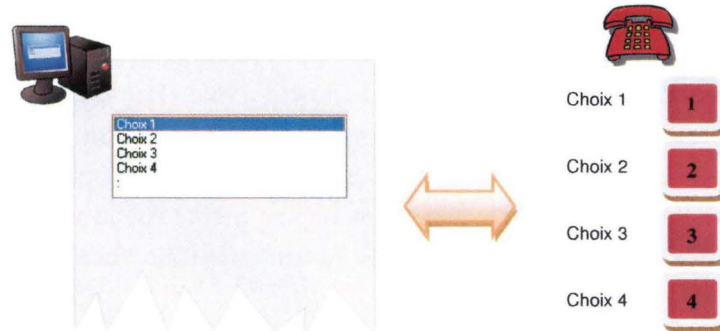


Figure I.13 : Deux interfaces pour deux modalités d'interaction différentes

III.2.3 Types de composition

L'architecture du système est construite pendant la conception et avant l'exploitation du système. Le changement au niveau de l'environnement d'exécution entraîne la nécessité de mise à jour de la composition du système afin de s'adapter au contexte d'usage. La modification de l'architecture de l'application peut se faire en ajoutant de nouvelles instances de composants, en supprimant des instances déjà existantes, en changeant les interconnexions entre les instances, ou en réglant les paramètres qui déterminent le comportement du système. Deux types de composition sont distingués : la composition statique et celle dynamique (cf. Figure I.14).

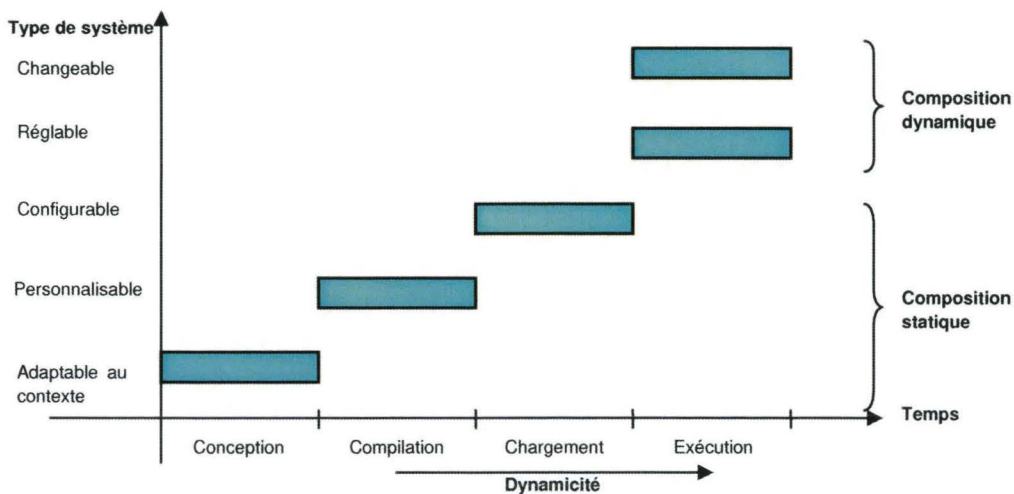


Figure I.14 : Classification de la composition de systèmes en fonction du temps de la composition ou la recomposition [McKinley et al., 2004a]

III.2.3.1 Composition statique

Dans le cadre d'une composition statique, l'architecture du système est construite en sélectionnant des composants adéquats à l'environnement d'exécution envisagé. Cette composition ne peut être modifiée, dans le but d'adaptation aux nouvelles situations que par le concepteur, à la phase de conception.

Généralement si un système est développé pendant la conception, aucune adaptabilité ensuite ne peut être appliquée au système *pendant l'exécution* – le changement au niveau de la composition du système ne peut être effectué qu'à la conception. De façon limitée, le changement de composition peut être mis en application au moment de la compilation du système et avant sa distribution vers le dispositif cible. Par exemple, les langages de programmation orientés aspect tels qu'AspectJ [Kiczales *et al.*, 2001] permettent de sélectionner des aspects pour le système en cours de sa compilation, en fonction du contexte d'usage prévu. Différents aspects pour une même fonction pourraient prendre en considération des propriétés spécifiques d'un environnement de travail ou des caractéristiques spéciales d'une plate-forme.

Il existe des systèmes personnalisables au moment de la compilation qui exigent seulement à être recompilés à nouveau pour chaque adaptation à un nouveau contexte. Il existe aussi des systèmes ayant la capacité de s'adapter pendant le chargement du système (chargement des classes). La composition de ce type des systèmes est appelée la *composition en temps de chargement*. L'avantage de ce type de composition est que des décisions peuvent être prises pendant le chargement du système et peut-être à l'exécution (de nouvelles classes peuvent être chargées à l'exécution en cas de nécessité). Les décisions peuvent être prises sur des unités algorithmiques, des classes, ou des composants à choisir en fonction du contexte actuel. Dans ce cas, des algorithmes et des composants de différents choix doivent être totalement intégrés au code distribué, pendant la compilation, et doivent être par la suite accessibles facilement dans les ressources du système. Par exemple, dans le cas de la machine virtuelle de Java (JVM) des classes sont chargées lorsqu'elles sont appelées à l'exécution. Dans ce cas JVM pourrait utiliser une autre classe à la place d'une classe déjà définie pour un tel contexte, dans le cas où les conditions de l'environnement de travail incitent à privilégier telle classe plutôt qu'une autre. Dans la Figure I.15, JVM a le choix d'appeler la classe *Classe01_C1* pour le contexte prédéfini *C1*, ou d'appeler la classe *Classe01_C2* pour le contexte *C2*.

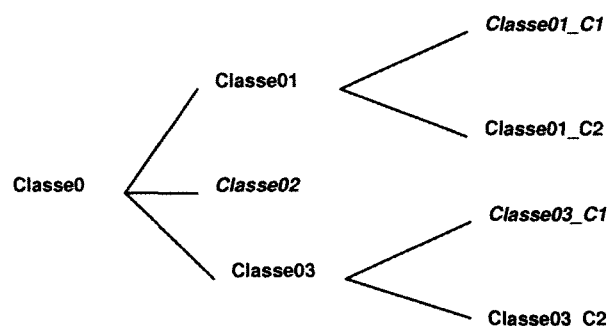


Figure I.15 : Possibilité de choix des classes à l'exécution par JVM

La composition en temps de chargement permet donc à la composition statique de gagner en dynamique. Lorsque le système requiert le chargement d'un nouveau composant, la logique de décision choisit parmi une liste de composants déjà inclus dans le code distribué, en prenant en compte les besoins du système, les conditions actuelles d'usage, et la capacité du composant choisi. Par exemple, un système peut avoir la capacité de choisir des composants à l'exécution en fonction de l'espace d'affichage disponible sur la plate-forme actuelle. D'autres approches de temps de chargement sont basées sur l'idée de modifier dynamiquement la classe elle-même lors de son chargement. Par exemple, *kava* (reflective Java based on bytecode rewriting) [Welch et Stroud, 2000] fournit un chargeur de classe qui re-décrit les classes pendant qu'elles sont chargées afin d'augmenter la possibilité de contrôler et de déboguer le système à l'exécution sans nécessité de revenir à la conception.

McBryan et Gray ont introduit la notion de fonction d'évaluation [McBryan et Gray, 2008]. Ils offrent une approche permettant de combiner un éventail de techniques permettant de (re)configurer l'IHM, avec la possibilité de choisir la manière de les combiner. Pour cela ils se basent sur la notion de possibilité de configuration de l'IHM, sur des fonctions d'évaluation (applicables à des ensembles de possibilités de configuration), et sur des approches utilisées pour paramétrer les fonctions et combiner les résultats.

III.2.3.2 Composition dynamique

Les approches les plus souples sont celles qui mettent en application la composition à l'exécution : des unités algorithmiques et structurales peuvent être remplacées ou prolongées pendant l'exécution sans arrêter et redémarrer le système. Nous distinguons deux types d'approches dont le noyau fonctionnel du système peut être modifié ou non [McKinley *et al.*, 2004b].

- Le logiciel réglable (*Tunable Software*) interdit la modification du code distribué. Par contre le logiciel réglable permet de répondre aux changements environnementaux, tels que les conditions dynamiques produites dans un environnement mobile en redéfinissant les paramètres du système. Un exemple concret est le modèle d'objet de fragment utilisé dans AspectIX [Geier *et al.*, 1998], qui permet à une application CORBA d'être réglée à l'exécution.
- En revanche, un logiciel changeable (*Mutable Software*) permet le changement d'un processus dans un système à l'exécution, en permettant à ce système d'être dynamiquement recomposé. Par exemple, dans la plate-forme middleware OpenORB [Blair *et al.*, 1998] tous les objets (dans le middleware² et dans le code

² Un middleware est une couche de services séparant le noyau fonctionnel des systèmes, du système d'exploitation [Blair *et al.*, 1998]. Comme le rôle traditionnel du middleware est de cacher les ressources et l'hétérogénéité de la plateforme, du noyau fonctionnel des applications, le middleware est utilisé, par

du système) ont des interfaces réfléchissantes. A l'exécution virtuellement n'importe quelle modification peut être apportée à n'importe quel objet (changer l'interface, l'implémentation, etc.). Tandis que dans la plupart des cas cette flexibilité doit être contrainte pour assurer l'application de l'adaptation au système.

III.2.4 Approches à base de composants

Comme nous l'avons vu précédemment l'idée d'adopter une approche à base de composants dans l'ingénierie des systèmes d'information vient de la volonté de mettre en œuvre des propriétés de réutilisation, de coopération et d'interopérabilité. Dans ce qui suit, nous présentons brièvement trois méthodes largement reconnues à savoir *Catalysis* [D'Souza, 2003], *Symphony* [Hassine *et al.*, 2002b] et *Select Perspective* [Allen et Frost, 1998].

III.2.4.1 *Catalysis*

Catalysis [D'Souza, 2003] est une méthode de conception d'applications à base de composants abstraits. La méthode est basée sur le principe de raffinement de types (pour spécifier la sémantique d'un objet), et de collaboration entre composants. Dans cette méthode, un composant est "un paquetage cohérent de logiciel qui peut être développé indépendamment et livré comme une unité, et qui définit les interfaces par lesquelles il peut être composé avec d'autres composants pour fournir et utiliser des services". Le processus de développement que propose *Catalysis* se décompose en quatre grandes étapes : l'analyse des besoins, les spécifications du système, la conception de l'architecture puis la conception interne des composants.

III.2.4.2 *Symphony*

Symphony est une démarche de développement logiciel élaborée et commercialisée par la société Umanis avec la collaboration de l'équipe SIGMA du laboratoire LSR-IMAG [Hassine *et al.*, 2002b]. Le modèle de composants proposé par la démarche *Symphony* est plus particulièrement axé sur les composants métiers. L'objectif est l'utilisation systématique de composants tout le long du cycle de vie et leur identification dès la phase d'expression des besoins. La démarche *Symphony* s'appuie sur le langage UML (Unified Modeling Language) [Booch *et al.*, 1998] et sur les bonnes pratiques de développement orienté objet [Jacobsen et John, 2000]. Elle se présente sous forme d'un guide méthodologique offrant une solution basée sur l'utilisation de composants (cf. Figure I.17). Dans *Symphony*, l'architecture des composants métiers est une structuration inspirée de la technique CRC (Classe-Responsabilité-Collaboration) [Wirfs-Brock *et al.*, 1990]. Un composant métier est modélisé par un paquetage composé de trois parties (Figure I.16) :

- une partie contrat avec l'extérieur (*ce que je sais faire*),

[Fitzpatrick *et al.*, 1998], [Redmond *et al.*, 2002] et [Klefstad *et al.*, 2002], comme un endroit logique pour mettre des mécanismes relatifs à l'adaptation des systèmes au contexte d'usage.

- une partie structurale (*ce que je suis*),
- et une partie collaboratrice (*ce que j'utilise*).

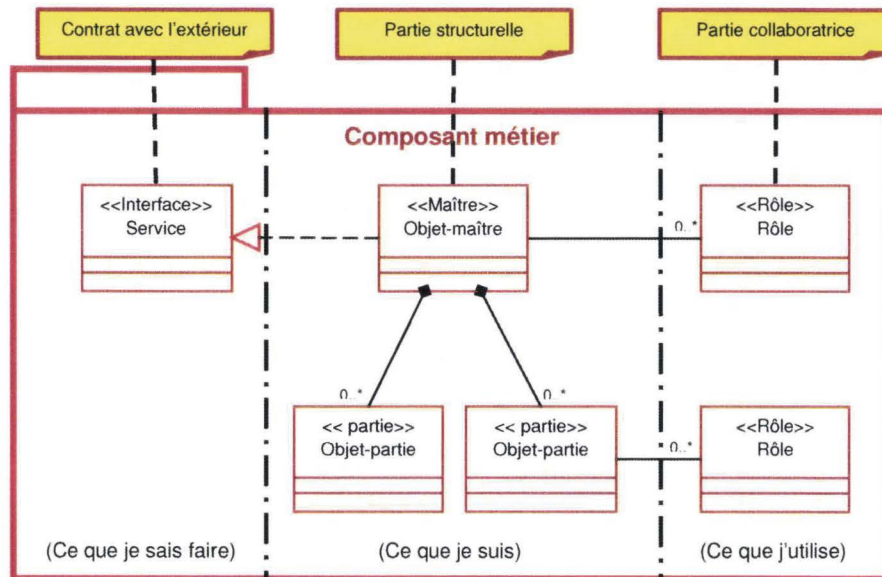


Figure I.16 : Architecture d'un composant métier Symphony [Hassine et al., 2002b]

Les trois parties du composant sont respectivement représentées par cinq types d'objets stéréotypés par :

- **Maître** : est l'objet principal du composant pour lequel les services sont réalisés. Il est qualifié de hiérarchiquement supérieur et est identifiable par tout acteur externe.
- **Partie** : est un objet complémentaire à l'objet « Maître ». Il est identifié par son attachement à l'objet « Maître » auquel il est relié par une relation de composition.
- **Rôle** : représente un objet fournisseur de l'objet « Maître ». Tous les services des autres composants sont appelés au travers d'un objet « Rôle ». Un objet « Rôle » est relié à son maître (ou à une partie) par une relation de composition.
- **Interface** : est l'objet représentant les services offerts par le composant. Il supporte les opérations représentant d'une part, les cas d'utilisation du composant et d'autre part, les services attendus par les rôles qu'il joue dans les autres composants.

Symphony propose également plusieurs relations inter-composants. Seule la relation d'utilisation, utile pour notre étude, est décrite ici. Cette relation permet de décrire le rôle joué par un composant pour un autre en termes des services attendus par ce dernier. Elle met en œuvre une classe *Rôle* dans le composant client relié au composant fournisseur (via son interface) par une dépendance stéréotypée « utilisation ». La classe *Rôle* peut définir en son sein un ensemble de propriétés dérivées, images de propriétés de la classe *Maître* cible ainsi que des propriétés intrinsèques.

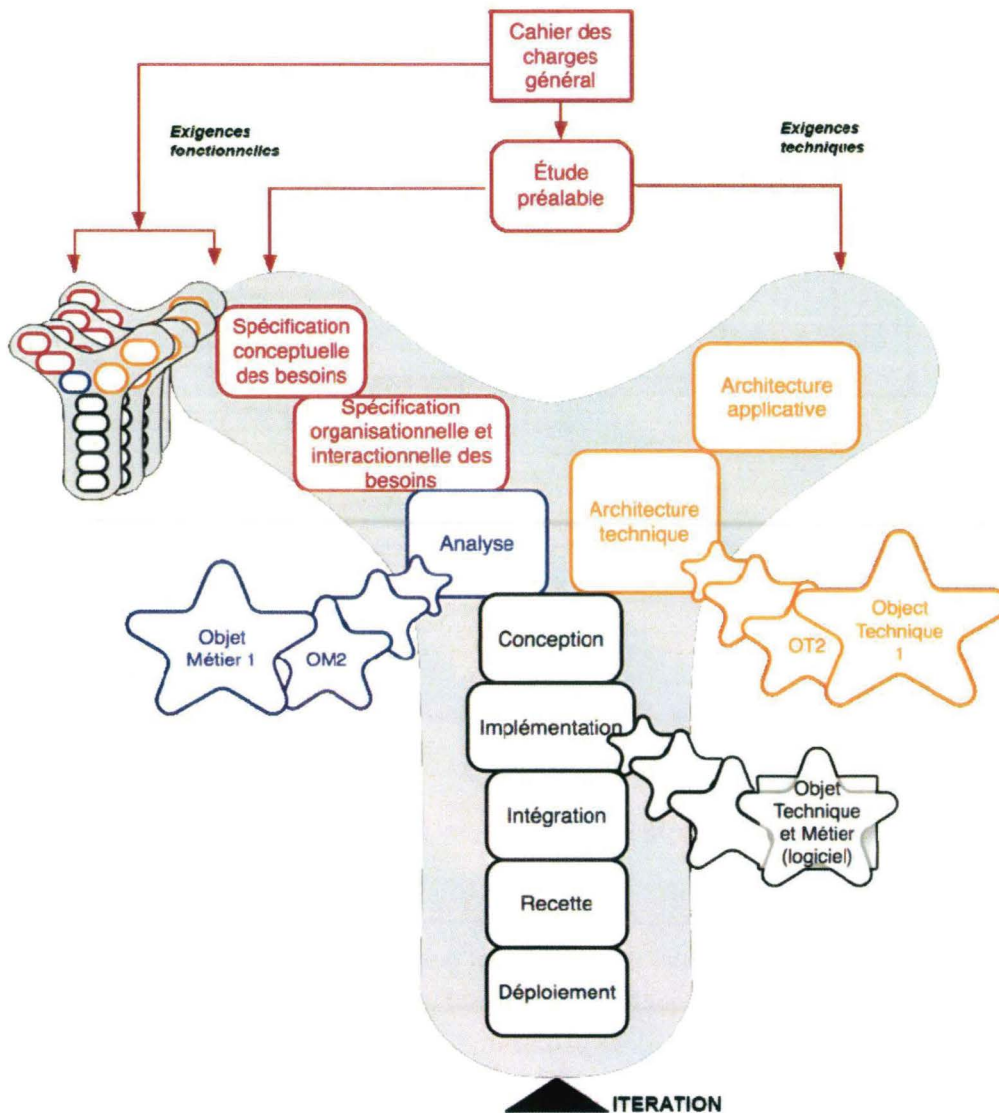


Figure I.17 : Cycle de vie en Y de la démarche Symphony [Hassine, 2005]

III.2.4.3 Select Perspective

Select Perspective [Frost, 1995], cité dans [Allen et Frost, 1998], se compose de phases de recherche, de sélection, d'adaptation et d'intégration de composants réutilisables. Elle se base sur une architecture logicielle orientée métier. *Select Perspective* fournit la modélisation des composants orientés métier, des composants de conception, et du déploiement.

Le processus de développement est scindé en deux processus :

- Le processus solution vise le développement de solutions en termes de services utilisateurs pour couvrir un maximum de besoin des utilisateurs ;
- Le processus de composant vise à développer des composants supportant le métier et/ou les services plus spécifiques.

III.2.4.4 Conclusion sur les approches à base de composants

Toutes les méthodes présentées adoptent un processus de développement itératif et incrémental. Ce type de processus permet en principe de déboucher sur des applications de qualité grâce au contrôle du temps, des coûts et des risques [Barbier *et al.*, 2002]. Les processus de développement proposés ont la particularité d'être flexibles afin de s'adapter aux besoins d'un projet spécifique ou d'une organisation. Elles tentent d'intégrer les composants métier dès le début du cycle de vie. Elles visent également à être indépendantes de la plateforme de développement. Elles proposent généralement une architecture de composants.

III.2.5 Conclusion sur la composition

Divers types de composants peuvent être utilisés séparément ou en collaboration avec d'autres types de composants, pour construire des systèmes interactifs. Des recherches ont été effectuées afin de charger et décharger des composants dynamiquement à l'exécution.

Nous allons utiliser dans notre méthode deux types de composants (composants métier et composants de présentation) en définissant une collaboration entre eux. Cette collaboration servira à adapter l'IHM en fonction du contexte d'usage. Dans notre démarche décrite dans le chapitre III, nous associons à une ou plusieurs tâches du système, un composant mixte composé à la fois de sa partie métier (logiciel) et de sa partie interactive dédiée aux interactions avec l'utilisateur, en nous inspirant de [Szyperski, 2002].

IV Nouveaux systèmes interactifs adaptatifs et mobiles

Les dénominations "sensible au contexte", "adaptable au contexte" ou "qui prend en compte le contexte", proviennent toutes du terme anglais "context-aware computing" qui a été introduit pour la première fois par Schilit et Theimer [Schilit et Theimer, 1994]. Ces auteurs ont décrit un système interactif sensible au contexte, ce système utilisant des informations sur la localisation et l'identité des voisins de l'utilisateur et des changements au niveau des informations utilisées. Les travaux [Brown *et al.*, 1997], dans ce domaine, utilisent ces informations et d'autres informations comme le temps, les caractéristiques physiques de l'environnement, le niveau de connaissance de l'utilisateur, etc. Le contexte est une des informations qui peut être utilisée pour caractériser la situation d'une entité (comme une personne, un environnement ou une plate-forme) [Abowd *et al.*, 1996]. En revanche l'informatique contextuelle (*context-aware computing* en anglais) est l'utilisation du contexte afin de fournir des informations et des services à l'utilisateur. La Figure I.18 illustre un exemple d'un système sensible au contexte. Selon [Dey, 2001], l'informatique contextuelle nécessite :

- La représentation des informations et des services à l'utilisateur ;
- L'exécution automatique de service(s) ;
- La capture des informations contextuelles.

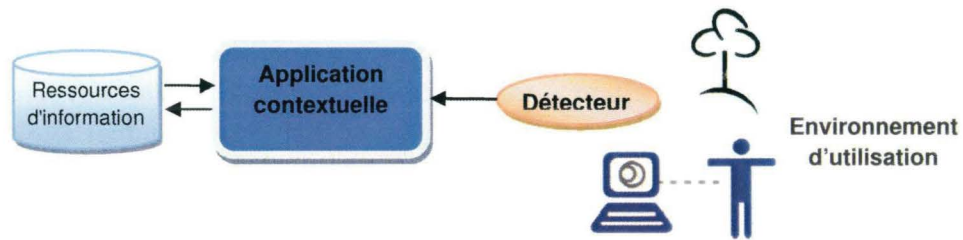


Figure 1.18: Exemple de système sensible au contexte

IV.1 Context-Awareness

Informatique pervasive ou ubiquitaire est le terme pour décrire le troisième âge de l'informatique dont la réalité augmentée en concevant et en développant de nouvelles modalités d'interaction [Kindberg et Fox, 2002] (en sortie, du système vers l'utilisateur, comme en entrée, de l'utilisateur vers le système) et de formes d'interaction multimodale [Schmidt, 1999]. L'informatique pervasive est liée à l'explosion des réseaux de communication sans fil [Pottie et Kaiser, 2000], aux avancées en terme de miniaturisation des dispositifs électroniques surtout celle des microprocesseurs, et aux progrès en matière d'accès aux informations partout et à tout moment [Abowd *et al.*, 2000].

L'informatique pervasive est née de l'idée de Marc Weiser des laboratoires Xerox, qui, le premier, a travaillé sur ce sujet [Weiser, 1991]. Elle est parfaitement à l'opposé de la notion de réalité virtuelle qui tente de mettre l'informatique autour de l'homme ; en effet, l'informatique pervasive insiste sur l'intégration de l'informatique partout au monde réel [Mackay *et al.*, 1993]. De plus, les recherches en informatique embarquée cherchent également à nous faire quitter la métaphore du bureau comme porte d'entrée du monde de l'information [Hess, 2002].

Au niveau de l'informatique pervasive, on a quatre types d'interaction. Le premier correspond aux relations entre les hommes, le deuxième aux relations entre l'homme et la machine, le troisième aux relations entre les machines, et les relations entre l'homme et l'environnement.

IV.2 Exemple informel³

Une entreprise de distribution d'eau potable dans la région Nord-Pas du Calais, possède plus de deux cents agents mobiles effectuant des interventions sur le réseau et/ou chargés de relever les compteurs. L'entreprise vient de décider de remplacer l'ancien système installé sur les terminaux portables de leurs agents, par un système embarqué qui sera inédit pour la planification des interventions clientèles et réseaux. Ce système doit

³ Le scénario de l'exemple est adapté d'un article de la Voix du Nord, n° janvier 2007, intitulé "Quand planification rime avec organisation".

permettre d'optimiser les déplacements des agents et d'agir rapidement en cas de besoin urgent. Les terminaux affichent aux agents la liste des interventions demandées et peuvent aussi récupérer des informations propres à chaque client chez qui ils ont à effectuer une intervention. Les clients peuvent poser aussi des questions aux agents sur leur consommation par exemple grâce au système qui peut récupérer des informations concernant le client. Les agents peuvent établir un compte-rendu de visite, ou aussi relever le compteur directement sur leur terminal informatique qui transfère ces informations dans la fiche client en base de données centrale.

On s'intéresse ici à l'interface graphique du système spécifié ci-dessus. Ce système est orienté opérateurs nomades, doit posséder une interface graphique ayant la capacité de s'adapter de manière dynamique au contexte d'usage (Plate-forme, Utilisateur, et environnement) (cf. Figure I.19).

- *Contexte lié à la plate-forme* : le système sera installé sur plus de deux cents terminaux de différentes générations. Une partie des terminaux comporte un clavier physique ; les autres utilisent seulement un clavier virtuel. La taille de l'écran est un facteur essentiel pour l'interface graphique ; la taille de l'écran est variable selon la génération du terminal : 220*120 ou 320*240 pixels.
- *Contexte lié à l'utilisateur* : les utilisateurs (agents de l'entreprise) peuvent utiliser des terminaux différents selon la disponibilité, les tâches à accomplir, et par rapport à leurs besoins. Suite à la phase d'authentification, l'information concernant l'agent sera récupérée du serveur. De plus chaque opérateur peut avoir des préférences stockées dans la base de données centrale, récupérées lors de la connexion de l'agent.
- *Contexte lié à l'environnement* : les agents travaillent dans des environnements différents : les caractéristiques de l'environnement changent fréquemment. Parfois la luminosité augmente (il sera nécessaire d'éclairer l'arrière-plan et foncer les textes), ou le niveau de bruit s'élève, etc.

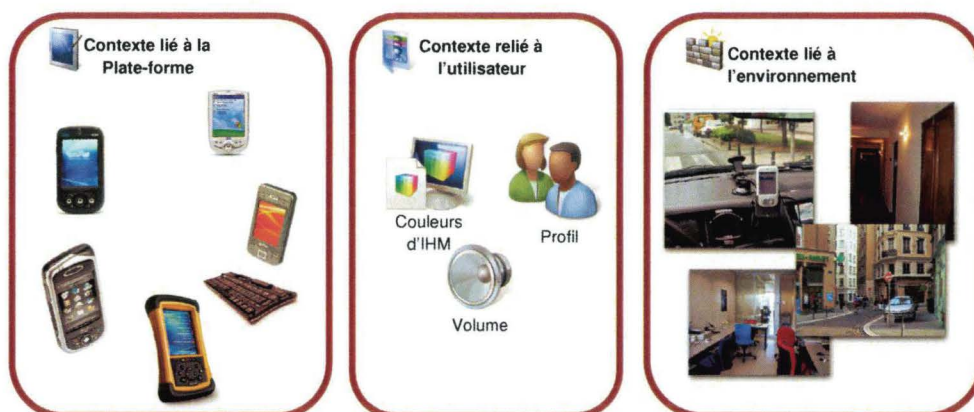


Figure I.19 : Contexte d'usage

IV.3 Recherche de nouveaux types d'IHM

Avec l'apparition de nouveaux médias, de nouveaux moyens de communication, de nouveaux terminaux, les avancées des réseaux et les progrès en miniaturisation, l'utilisateur est considéré comme mobile, évoluant dans un environnement varié et recourant, de manière opportuniste, à des plates-formes d'interaction diverses. Tout cela a créé de nouvelles considérations et de nouveaux enjeux à satisfaire. Des recherches portent sur de nouveaux types d'IHM, qu'elles soient dites sensibles au contexte, ou encore plastiques. Ce type d'interfaces doivent avoir la capacité de s'adapter de manière dynamique à son contexte d'usage en découvrant et utilisant des informations contextuelles, telles que la localisation de l'utilisateur, la date et l'heure, la proximité d'autres utilisateurs, les ressources disponibles, les caractéristiques de la plate-forme cible, les conditions ambiantes, etc. Durant l'exécution, il s'agit de répondre dynamiquement aux modifications contextuelles tout en respectant l'utilisabilité du système selon les premières définitions proposées par ces auteurs.

IV.3.1 Plasticité et adaptation

La plasticité d'une interface [Calvary *et al.*, 2001a ; 2001b ; 2003] dénote sa capacité à s'adapter aux variations du contexte d'usage en termes d'utilisateur, de plate-forme et/ou d'environnement dans le respect de son utilisabilité [Nielsen, 1993].

Nous avons principalement deux solutions pour accroître la capacité d'adaptation de l'IHM :

- **Adaptation statique** (ou manuelle) : elle consiste à préparer plusieurs versions du même système avant la distribution vers la plate-forme cible, plus précisément plusieurs versions des ressources ; par exemple, plusieurs versions des ressources textuelles sont préparées de façon plus moins détaillée en fonction de l'âge de l'utilisateur, et en plusieurs langues ; ensuite à l'exécution ces ressources prédéfinies sont utilisées par des terminaux standards dans des contextes d'usage prédéfinis. Des transformations doivent être appliquées sur ces ressources afin d'être exploitées pour un nouveau contexte d'usage. Les transformations peuvent être effectuées par l'utilisateur ou par un expert. Par exemple, [Gajos *et al.*, 2007] proposent l'approche *SUPPLE++* permettant de générer l'IHM selon une approche dirigée par les modèles : l'IHM est dans un premier temps spécifiée en fonction de plusieurs caractéristiques de l'utilisateur ; puis celui-ci passe un test, ensuite l'IHM est personnalisée selon les capacités (par ex. visuelles) de l'utilisateur.
- **Adaptation dynamique** : celle-ci consiste à effectuer, contrairement à la précédente, un certain nombre de transformations sur la ressource au cours de l'exécution du système. Un système d'adaptation automatique intercepte les requêtes des utilisateurs et effectue ces transformations suivant le contexte d'usage changé, en permettant aussi de minimiser le coût de développement et de maintenance.

La Figure 1.20 illustre l'espace problème de la plasticité [Calvary *et al.*, 2006] qui se caractérise comme suit :

- Au niveau de la modalité, le remodelage *intra-modal* permet de transférer l'IHM vers une autre modalité de même type (par exemple, graphique vers graphique) ; le remodelage *inter-modal* permet à l'IHM de migrer vers une modalité d'une autre famille (par exemple graphique vers vocal), enfin, dans le remodelage *multi-modal*, il s'agit de la combinaison entre des modalités ;
- Au niveau du contexte d'usage, on s'intéresse aux caractéristiques de la plate-forme, de l'utilisateur, et/ou de l'environnement (cf. aussi §IV.3.2).
- Au niveau temporel, l'adaptation reflète le caractère statique ou dynamique. A la différence de la version statique où les contraintes sont connues dès la conception et les IHM créées avant exécution, la version dynamique élabore des IHM, à la volée, au fil des nouvelles contraintes ;
- Au niveau de l'apprentissage, l'IHM plastique pourrait prendre en compte des nouveaux états du contexte d'usage, améliorer des règles d'adaptation, ou encore améliorer la connaissance lié à la conception de l'IHM ;
- Au niveau de la granularité de l'adaptation, cette adaptation se déroule au niveau des interacteurs (par exemple, un bouton radio dans une fenêtre), à celui de l'espace de travail (un ensemble de tâches logiquement connectées), par exemple modification d'une fenêtre, ou au niveau de toute l'IHM.

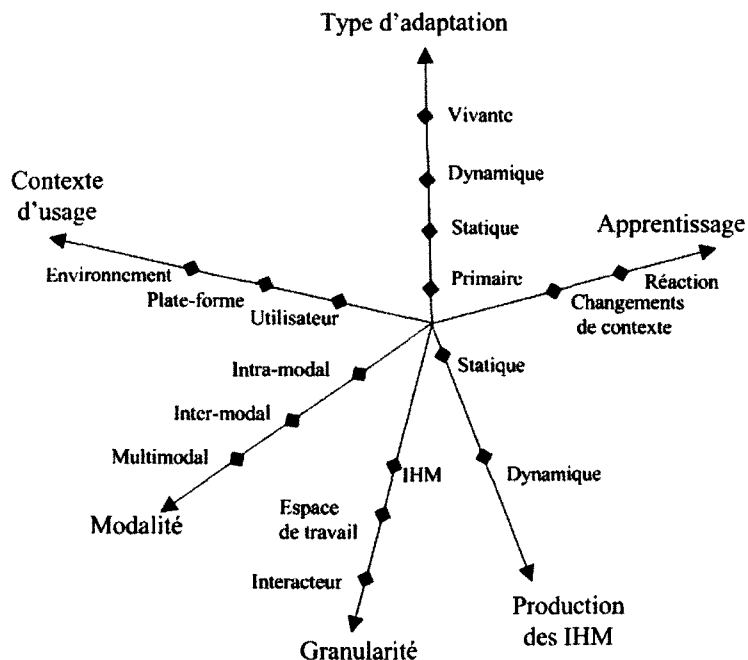


Figure 1.20 : Espace problème de la plasticité (adapté et simplifié de [Calvary *et al.*, 2006])

Les cas élémentaires de plasticité sont ceux où l'adaptation ne s'effectue que sur un seul type de contexte (en particulier le contexte lié à la plate-forme). En dehors de ces cas

élémentaires, la plasticité introduit des cas plus complexes d'adaptation à plusieurs contextes simultanément, de manière statique ou dynamique. Par exemple le cas de l'adaptation à plusieurs plateformes d'interaction pour une même tâche introduit la problématique de la continuité de l'interaction (exemple du passage d'un PC à un PDA avec souhait de continuer à travailler sur la même tâche).

IV.3.2 Contexte d'usage

Le contexte d'usage du système contient des informations à considérer pendant la conception et l'exécution du système. La définition la plus couramment adoptée par les équipes travaillant dans le domaine de l'adaptation des systèmes interactifs identifie le contexte par le triplet « Plate-forme, Utilisateur, Environnement ». Par conséquent dans ce domaine trois catégories d'informations contextuelles peuvent être distinguées [Dey, 2000] :

- Celles relatives à la plate-forme (qui se décline en deux sous catégories : la plate-forme physique (matérielle) et la plate-forme logique (logicielle)), telles que ses caractéristiques (processeur, mémoire, équipements périphériques), les réseaux de connexion, la taille de l'écran d'affichage, et les outils d'interaction disponibles.
- Celles relatives à l'utilisateur, telles que son profil, son niveau professionnel, sa localisation physique, son activité courante (loisir, travail), ses préférences et ses habitudes, etc.
- Les informations relatives à l'environnement physique et social, comme le niveau de bruit, la luminosité, la température, et les informations temporelles.

Les informations contextuelles sont stockées et actualisées lors de la capture du contexte. Celles-ci sont perçues et envoyées régulièrement au système pour évaluer l'utilisabilité de l'IHM. En cas de changement au niveau du contexte, une adaptation est lancée afin de préserver l'utilisabilité.

IV.4 Conclusion sur les nouveaux systèmes interactifs adaptatifs et mobiles

Avec l'informatique pervasive, "la plasticité des interfaces est devenue une nécessité" [Calvary et Coutaz, 2002a]. Cette dernière permet d'adapter l'IHM au contexte d'usage (plate-forme, utilisateur et environnement) en préservant l'utilisabilité de l'IHM. Beaucoup de travaux portant sur un tel type d'IHM essayent de définir plus précisément la notion de plasticité, et de proposer des méthodes et des approches orienté multi-modalité en se basant sur des mécanismes et des outils existants et aussi sur des nouvelles technologies.

V. Conclusion

Nous avons présenté, dans ce chapitre, différentes architectures de systèmes interactifs ; les principes de base de différents modèles ont été exposés. Leur objectif est de fournir des

stratégies de décomposition fonctionnelle et/ou structurelle des systèmes interactifs. Tout d'abord nous avons présenté les architectures centralisées qui sont historiquement les premiers modèles d'architecture à avoir vu le jour. Elles décomposent un système interactif en plusieurs couches, de celle responsable des interactions directes avec l'utilisateur (interface homme-machine) jusqu'au noyau fonctionnel (application). Ensuite les architectures réparties (multi-agents) ont été présentées. Elles décrivent un système interactif en un ensemble d'agents. Le rôle et la composition des agents varient d'une architecture à une autre. Enfin, nous avons terminé cette partie avec la présentation des architectures hybrides, tentant de tirer profit des avantages des modèles centralisés et répartis.

Ensuite, les patrons de conception, les compositions et les composants de systèmes interactifs ont fait l'objet de la deuxième partie. Les patrons de conception offrent des solutions efficaces et éprouvées à des problèmes de conception et reflètent les meilleures pratiques de conception. Des chercheurs ont travaillé sur cette notion en proposant des catégories de patrons applicables à différentes phases de la conception des systèmes. Ils ont proposé des langages de patrons. Par exemple, Gamma et ses collègues [Gamma *et al.*, 1995] ont adopté une représentation uniforme et structurée pour les patrons, afin de faciliter et favoriser l'utilisation des patrons de conception par les développeurs de logiciels. La notion de patron de conception sera intégrée dans notre démarche, qui sera proposée dans le chapitre III, afin de réutiliser des expériences en conception des systèmes plastiques, et d'avoir des solutions applicables à l'exécution. Ensuite, les composants ont été présentés. Afin d'éclaircir la notion de composant, telle qu'elle est vue au sein de la communauté scientifique, nous avons défini les notions de réutilisation, de composant, et d'approche à base de composants. Enfin, nous avons décrit la notion de composition des systèmes, celle-ci pouvant être statique ou dynamique. En cas de composition statique, le système est développé pendant la conception, cependant aucune adaptabilité ensuite ne peut être appliquée au système pendant l'exécution. La composition dynamique permet de changer la structure du système pendant l'exécution, sans arrêter et redémarrer le système. Des composants d'IHM, des algorithmes peuvent être supprimés, ajoutés ou remplacés à l'exécution. La technique de la composition lors d'un chargement permet à la composition statique de gagner en dynamique. Cette technique donne la possibilité de choisir entre plusieurs versions d'une même classe déjà préparées pour différents contextes d'usage. Nous nous intéresserons par la suite à la composition dynamique de système afin de supporter l'adaptation de systèmes plastiques à l'exécution.

Enfin, nous nous sommes penchés sur une nouvelle génération de systèmes interactifs adaptables, placés dans un environnement non homogène et changeable à tout moment. Nous avons défini la plasticité de l'IHM avec ses axes et son espace problème, le contexte d'usage auquel l'IHM doit s'adapter. La plasticité consiste à adapter les IHM au contexte d'usage en préservant leur utilisabilité. La plupart des chercheurs du domaine considèrent que le contexte d'usage est identifié par le triplet « Plate-forme, Utilisateur, Environnement ».

Dans le chapitre suivant, différentes méthodes et outils orientées multiplateformes sont passées en revue. Ceux-ci permettent de générer plusieurs versions d'une IHM à partir d'une seule spécification. Ensuite, nous aborderons des méthodes orientés contexte ou plasticité avec un souci de comparaison globale et critique.

Chapitre II

**Méthodes et langages pour la spécification et
la génération d'IHM multiplateforme et/ou
orientée contexte**

I. Introduction

Nous avons présenté dans le précédent chapitre, les architectures de systèmes interactifs, et la nouvelle génération d'IHM qualifiée de plastique qui s'adapte à l'environnement de travail changeable et mobile dans le respect de son utilisabilité. Dans ce chapitre, nous présenterons des méthodes et des langages orientés multiplateformes qui ont l'objectif de spécifier et générer l'IHM multicible. Le terme *multiplateformes* a été proposé pour la spécification abstraite de l'IHM indépendamment de la plate-forme cible, sachant qu'ensuite la version abstraite de l'IHM peut être transformée en version finale vers diverses plateformes. Cette version finale doit alors respecter les caractéristiques du support cible. Ensuite nous présenterons des approches à base de modèles orientées plasticité ; ces approches proposent des nouveaux modèles et de nouvelles étapes dans le cycle de développement de systèmes interactifs afin de permettre la génération d'IHM plastique. Les approches se différencient par leur capacité à établir une IHM capable de s'adapter automatiquement ou semi-automatiquement, par le type d'adaptation adopté, par les mécanismes d'adaptation ou encore par les outils qui supportent chaque approche. Avant de commencer à présenter les méthodes et les langages, nous définissons les critères et les mesures sur lesquelles notre comparaison sera établie.

II. Critères d'étude des méthodes et des langages étudiés

Nous allons étudier dans ce chapitre des méthodes et des langages de spécification et génération d'IHM multiplateforme et/ou orientée contexte. Afin de bien situer notre démarche proposée dans le chapitre III par rapport aux autres méthodes et langages, nous allons étudier chaque approche selon un ensemble de critères : modèle d'interface, outils, langages supportés, plates-formes supportées, contexte d'usage, adaptation, connaissance de conception, et technique d'apprentissage. Chacun des critères sera détaillé ci-après.

II.1 Modèle d'interface

Dans la Figure II.1, le modèle d'interface est une entité descriptive qui représente les caractéristiques de l'application du point de vue de l'interface, au sens large. Ceci inclut aussi bien les notions de dialogue (ce qui est permis à un instant donné), que de présentation. Le modèle peut globalement être décomposé en trois niveaux d'abstraction [Szekely, 1996] (cf. Figure II.1) :

1. Au plus haut niveau, on trouve les modèles de tâche, de domaine et d'utilisateur. Le modèle de tâche représente les tâches du système. Elles sont généralement décomposées en une hiérarchie de tâches et sous-tâches. Les tâches feuilles correspondent à des opérations directement réalisables avec l'application. Le modèle de domaine représente les objets manipulés par l'utilisateur, leurs relations, et les actions que le système peut effectuer sur ces objets. Le modèle de l'utilisateur propose une représentation détaillée du comportement de l'utilisateur vis-à-vis du modèle de tâche. Les tâches peuvent être supportées par des spécifications utilisées lors d'une adaptation. [Clerckx *et al.*, 2007] proposent une approche se focalisant

sur la spécification des tâches associées aux ressources requises, telles que celles du dialogue et de la présentation.

2. Le second niveau d'abstraction représente la structure générale de l'interface de l'application. Cette structure est décrite en termes d'interaction de bas niveau (sélection d'un élément dans un ensemble par exemple), d'unités de présentation (correspondant aux fenêtres), et d'éléments d'information (valeur, ou ensemble de valeurs du domaine, labels, constantes...). Cette structure définit de manière abstraite les informations à présenter à l'utilisateur et les dialogues autorisés pour interagir avec ces informations.
3. Le troisième niveau forme la spécification concrète de l'interface. Elle spécifie le rendu des informations du second niveau en termes d'éléments de boîte à outils (menu, boîte à cocher, ...). A ce niveau, des spécifications concernant le contexte d'usage (cf. §II.5) peuvent être prises en compte afin de rendre une IHM adaptée à son contexte d'usage envisagé.
4. L'IHM finale se trouve implicitement selon la Figure II.1, dans la boîte « application ». Il s'agit d'une version opérationnelle de l'interface qui peut être déployée sur le dispositif cible. Une méthode peut supporter une ou plusieurs plateformes (cf. §II.4).

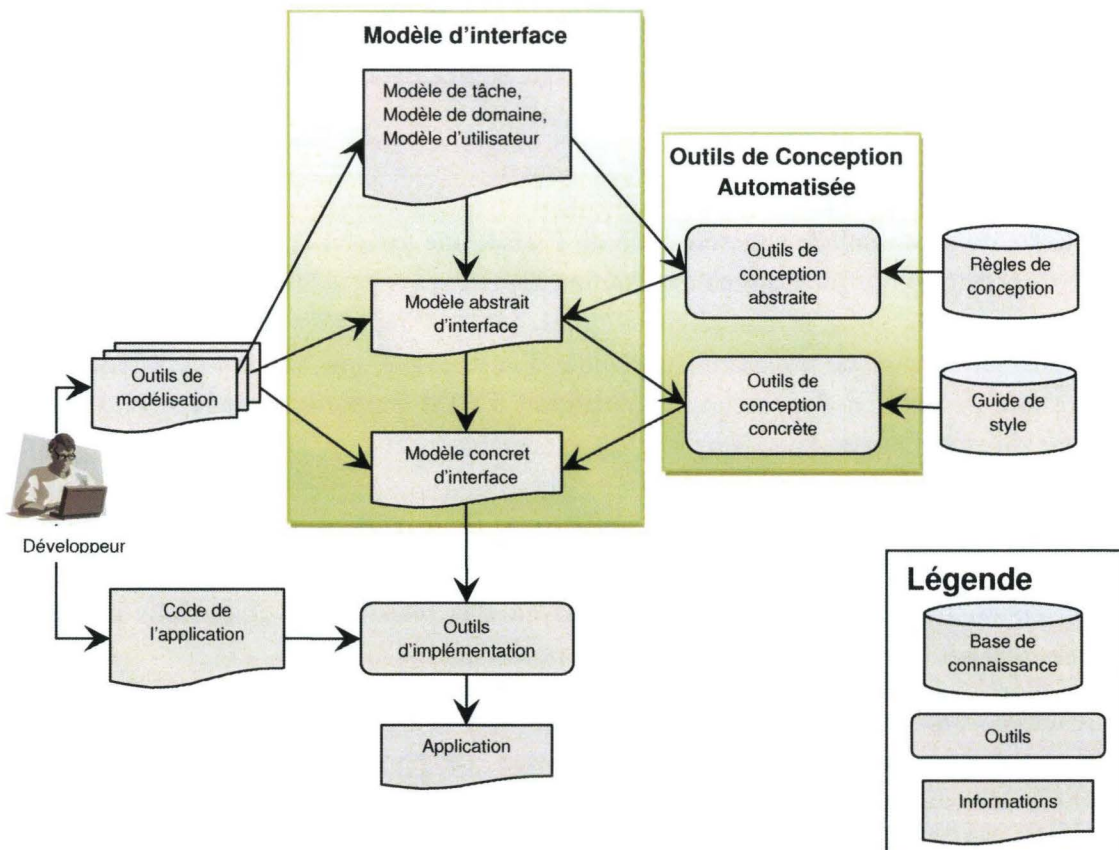


Figure II.1 : Environnement de développement d'interface à base de modèles (adapté par [Tabary, 2001] de [Szekely, 1996])

II.2 Outils

Certains langages et méthodes de spécification d'IHM sont supportés par des outils et des infrastructures logicielles afin de supporter les différentes étapes du cycle de développement d'IHM. Ces outils sont de trois types (cf. Figure II.1) :

- **Outils de modélisation** : ils assistent le concepteur du système dans une phase consistant à décrire des modèles à l'aide d'informations constitutives. Les outils de modélisation peuvent être des éditeurs de texte pour construire les spécifications textuelles comme dans UIML (sera détaillé dans §III.1), ou encore des éditeurs graphiques spécialisés comme dans UsiXML (cf. §III.8.2).
- **Outils de conception** : ils permettent au concepteur de construire des modèles à partir des spécifications. Les outils de conception automatique (ou semi-automatique) utilisent souvent une base de connaissances de conception et/ou des guides de style (par exemple, MOBI-D, cf. III.7). Il existe aussi des méthodes qui demandent explicitement la spécification de chaque niveau du modèle, et ne fournissent pas d'outil de conception automatique mais offrent généralement la possibilité de réutiliser les spécifications (voir par exemple, UIML, cf. §3.1).
- **Outils d'implémentation** : ils transforment la spécification concrète de l'IHM en une représentation qui peut être directement manipulée sur la plateforme cible. Ces outils peuvent être des moteurs de rendu (ex. Java, HTML, etc.), ou des générateurs capables de générer le code source dans un langage de programmation (ex. C++).

II.3 Langages supportés

A la fin du processus de concrétisation de l'IHM, une version de l'IHM peut être décrite dans un langage de programmation comme C++, Java, etc. L'avantage de cette idée est d'utiliser par la suite le compilateur et les outils d'implémentation et de distribution propres au langage de programmation cible. Par exemple, une version de l'IHM en Java peut être générée, si l'approche de génération d'IHM supporte le langage Java. Ensuite grâce au compilateur Java, une version exécutable peut être implémentée sur la plateforme cible (cf. ci-dessous).

II.4 Plateformes supportées

La version finale de l'IHM peut être envoyée à la plate-forme cible. Celle-ci doit être supportée par la méthode ou le langage de spécification d'IHM.

Le terme « *support d'une plate-forme* » indique que la version finale de l'IHM doit être en format exécutable sur la plate-forme supportée. De plus, l'IHM doit respecter le style propre du support cible, prenant en compte ses caractéristiques (taille de l'espace d'affichage, puissance de traitement de la machine, etc.). Certains méthodes et langages supportent un nombre limité de plateformes, dont la plupart doivent utiliser strictement les technologies du Web ou Java afin de réaliser la notion de multi-plate-forme.

II.5 Contexte d'usage

Le contexte d'usage (plate-forme, utilisateur et environnement) contient des informations à considérer pendant la conception et à l'exécution du système (cf. chapitre I, § IV.3.2).

Comme nous le verrons dans la suite du chapitre, des méthodes et langages de spécification d'IHM orientés contexte (ou exploitant la notion de plasticité) prennent en compte certains éléments du contexte d'usage à la conception (contexte d'usage prévu), et peuvent supporter l'IHM avec des mécanismes et des règles d'adaptation afin de répondre à certains changements contextuels à l'exécution.

II.6 Adaptation

Comme nous l'avons défini dans le chapitre I (cf. chapitre I, §IV.3.1), à l'adaptation correspond un ensemble de réactions ayant des effets sur l'IHM (ex. changements du style d'affichage, changement de la couleur d'arrière plan...). De plus l'adaptation peut être réalisée à la conception (statique) ou à l'exécution (dynamique). Dans le premier cas, l'IHM doit être transmise à l'étape de conception pour effectuer l'adaptation nécessaire vise à un ou des changements contextuels. Dans le deuxième cas, l'adaptation s'effectue pendant l'exécution de l'IHM sans nécessité de retour à la conception.

Les règles d'adaptation peuvent être spécifiées par le concepteur (règles d'adaptation statiques) ; ensuite ces règles peuvent être améliorées automatiquement ou de nouvelles règles peuvent être ajoutées (règles d'adaptation dynamiques) grâce à une technique d'apprentissage (si la méthode de génération d'IHM ou le langage de spécification d'IHM supporte la technique d'apprentissage, et fournit des mécanismes, avec l'IHM générée, pouvant supporter cette technique à l'exécution).

II.7 Connaissance de conception

Les méthodes peuvent posséder une base de connaissance de conception servant à guider les concepteurs pendant la spécification et la génération des IHM de manière consistante avec la plate-forme cible. Cette base peut contenir des informations conceptuelles concernant les styles des plateformes à respecter, et des règles ergonomiques. Deux grandes classes de règles peuvent être envisagées :

- les règles dépendantes des tâches : il est nécessaire de disposer d'informations sur l'utilisateur et/ou sa tâche et/ou le contexte pour pouvoir appliquer ces règles ergonomiques.
- les règles indépendantes des tâches : il n'est nécessaire de connaître, ni l'utilisateur, ni sa tâche, ni le contexte pour appliquer les règles ergonomiques. Ces règles génériques sont valides dans différentes situations.

La connaissance de conception est globalement contenue dans cinq types de sources [Vanderdonckt, 1994] : (1) les articles de recommandations proposant aux concepteurs,

des recommandations ergonomiques⁴ pour la conception des IHM, (2) les standards de conception qui normalisent les recommandations ergonomiques, (3) les algorithmes de conception ergonomique qui contrôlent la construction des IHM, (4) les guides de conception (ou guide de recommandations) qui contiennent un grand nombre de recommandations ergonomiques [Vanderdonckt, 1994], et (5) les guides de style spécifiques à un support [Simpson, 1999].

La Figure II.2 illustre des exemples de connaissances de conception. L'exemple en partie haute est organisé comme un tableau contenant une liste de composants de présentation avec leurs capacités d'affichage des informations à l'utilisateur ; par exemple, une règle ergonomique liée aux composants de type *Combobox* supportant la technique de recherche par la première lettre de mot, limite le nombre d'éléments (qui peuvent être intégrés dans ces composants) à cent. L'exemple en partie droite montre une variété de symboles utilisés dans les messages d'alerte, en fonction du système d'exploitation.

II.8 Apprentissage

L'apprentissage automatique est un des champs d'étude de l'intelligence artificielle [Russell *et al.*, 2002]. Elle consiste à améliorer la connaissance de conception, en extrayant de nouvelles expériences à l'exécution en apprenant par exemple du comportement de l'utilisateur. Dans le domaine de l'adaptation d'IHM, l'apprentissage peut jouer un rôle indispensable et complémentaire à l'adaptation. L'adaptation a pour objectif de préserver l'utilisabilité de l'IHM, alors que l'apprentissage devrait permettre de préserver l'utilité de l'adaptation.

La notion d'apprentissage peut être supportée en équipant l'IHM de mécanismes pouvant extraire des nouvelles connaissances. Celles-ci peuvent concerner des règles d'adaptation dans le cas de la plasticité. L'apprentissage sera présenté plus en détail, du point de vue de plasticité, dans le chapitre III.

⁴ Recommandation ergonomique : constitue un principe de conception et/ou d'évaluation à observer en vue d'obtenir et/ou de garantir une IHM ergonomique [Vanderdonckt, 1999b]

Controls	Number of choices	Types of choices	Shown as	Relative space used	Selection type
<i>Text-entry controls</i>					
Entry field (single line)	Not applicable	None	Alphanumeric	Low	Text
Entry field (multiple line)	Not applicable	None	Alphanumeric	Medium high	Text
Combo box (with first letter navigation)	100 or fewer	Variable settings choices or objects	Alphanumeric	High	Single choice in list and text in entry field
Combo box (with type ahead)	1,000 or fewer	Variable settings choices or objects	Alphanumeric	Medium high	Single choice in list and text in entry field
Drop-down combo box (with first letter navigation)	100 or fewer	Variable settings choices or objects	Alphanumeric	Low	Single choice in list and text in entry field
Drop-down combo box (with type ahead)	1,000 or fewer	Variable settings choices or objects	Alphanumeric	Low	Single choice in list and text in entry field
Spin button with an entry field (or spin box)	20 or fewer	Settings choices from an ordered list	Alphanumeric	Low	Single choice in list and text in entry field
<i>Non-text-entry controls</i>					
Push button	1 for each push button; 6 or fewer choices per field	Fixed action or routing	Alphanumeric, graphic	Low	Single

<i>Message Symbols</i>			
Classification	Symbol		
	Windows	UNIX	Swing
Information			
Warning			
Action			
Urgent Action			

Figure II.2 : Exemples de connaissances de conception⁵

II.9 Conclusion sur les critères d'étude

Nous venons de classifier et regrouper des critères de comparaison des méthodes et des langages de spécification et de génération des systèmes interactifs, s'appuyant sur les principes de multiplateforme et/ou de plasticité. Tout d'abord, chaque méthode ou langage peut définir le(s) modèle(s) d'IHM de son propre point de vue. Des outils peuvent supporter la mission du concepteur à différents niveaux de conception d'IHM. Certains langages et plates-formes peuvent être supportés par des méthodes ou langages au niveau final (implémentation d'IHM). Le support de la notion d'adaptation peut être varié, prenant en compte une gamme plus ou moins étendue de caractéristiques liées au contexte d'usage, avec ou sans possibilité d'améliorer la qualité des réactions en se basant sur une technique d'apprentissage.

⁵ Source : [http://www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/1392/\\$File/IBM_UIA.pdf](http://www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/1392/$File/IBM_UIA.pdf)

Nous étudions dans la partie suivante, plusieurs méthodes ou langages représentatifs, pour en faire ensuite une synthèse globale, selon les critères présentés ci-dessus.

III. Etude de méthode et langages pour la spécification et la génération d'IHM multiplateforme et/ou orientée contexte

Les approches orientées multicible consistent à spécifier une seule fois l'IHM. Ensuite, l'interface finale peut être générée, selon divers formats en fonction de la plate-forme cible, sans nécessité de changer le code ou le style d'affichage des composants d'IHM. La plupart des approches orientées multicibles sont des langages s'appuyant sur XML. Un des premiers avantages de ces langages est leur simplicité et la multitude des outils qui les supportent. Ils ont ensuite rendu possible la séparation du contenu d'un document de sa forme, mettant ainsi en œuvre les principes architecturaux basiques défendus par la communauté des chercheurs en Interaction Homme-Machine. Cette séparation a fait l'objet d'une première technique d'adaptation à travers la possibilité de recourir à des feuilles de style utilisées pour l'externalisation de la définition de la présentation d'un document (les couleurs, les polices, le rendu et d'autres caractéristiques) ; tout d'abord avec les CSS (Cascading Style Sheets) [Lie et Bos, 1999] associées à HTML puis avec XSL et les langages associés, proposés autour de XML. Dès lors, la création de différentes feuilles de style (qui contiennent les spécifications des adaptations souhaitées) conduit à la production de différents documents et cela pour le même contenu. Nous présenterons dans le cadre de ce travail, les langages de balisage multicibles UIML, AUIML, XIML, XForms, et Plastic ML.

Des approches à base de modèles ont évolué afin de prendre en compte le contexte d'usage dans un but de conception de systèmes interactifs qualifiés de sensibles à celui-ci [Van den Bergh *et al.*, 2004]. L'idée de ces approches est d'intégrer la capacité de spécifier partiellement ou intégralement les modèles correspondants au système, en fonction du contexte d'usage. Dans ces approches, certains modèles (comme les modèles de tâche, de dialogue ou de l'interface) ont évolué afin de supporter la notion d'adaptation tandis que d'autres modèles (comme le modèle de contexte, le modèle de transformation...) influencent cette adaptation. Le nombre de modèles pris en considération ainsi que le niveau de contextualisation d'un modèle (l'obtention d'un modèle propre au contexte) sont différents selon les approches [Tarpin, 2006]. Des projets de recherche existent depuis plusieurs années au niveau national (ex. projet Rainbow) et européen (Cameleon), visant à proposer des méthodes avec des outils supportant les notions inhérentes à la plasticité des IHM. Nous présenterons, dans ce chapitre, le projet Rainbow et l'outil AMUSINGs, le projet Cameleon avec ses approches Teresa et UsiXML, et la méthode MOBI-D, en tant qu'exemple d'approche de type MBD (Model-Based user interface Design) avec un point de vue multiplateforme. Nous présenterons également l'approche des *Comets* et celle de Thevenin.

III.1 UIML

UIML (User Interface Markup Language) [Abrams *et al.*, 1999 ; Abrams et Helms, 2004] est l'un des premiers langages basés sur XML pour décrire les interfaces multiplateformes. UIML est basé sur un méta-modèle illustré dans la Figure II.3. UIML a un vocabulaire permettant de décrire des IHM de manière générique, indépendamment de la plate-forme sur laquelle elles seront manipulées. De plus, ce vocabulaire définit les correspondances entre cette définition et les différentes implémentations qui en seront réalisées. L'IHM est décrite une fois, puis un interpréteur permettant d'interpréter l'interface dédiée à la plate-forme cible est utilisé. L'interpréteur traduit la description UIML de l'interface vers HTML, Java, WML ou VoiceXML (langage basé XML pour les supports vocaux). L'avantage d'UIML est qu'il permet de suivre les évolutions des plates-formes sans avoir besoin de changer le code source de l'interface. UIML permet de rendre l'écriture des IHM indépendantes de la plate-forme cible et donc de rendre celles-ci portables. Il est capable de projections vers différentes plates-formes (station, PDA, téléphone (modalité vocale) ...). Le langage permet également la séparation de ce qui est générique à l'interface de ce qui est spécifique aux supports. Une description UIML est composée de quatre parties :

- la partie « *head* » donne une méta-description du document ;
- la partie optionnelle « *template* » permet de décrire des fragments d'UIML réutilisables (cette partie permet de spécifier plusieurs facettes de présentation) ;
- la partie « *interface* » décrit les parties de l'interface utilisateur à savoir : la structure, le contenu, le style et les comportements ;
- la partie « *peers* » décrit la correspondance entre d'un côté les éléments du document UIML, et de l'autre les composants de l'interface finale et la logique d'application.

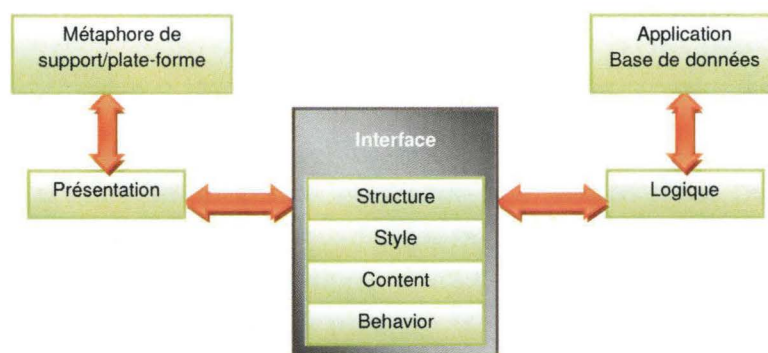


Figure II.3 : Modèles d'UIML

La Figure II.4 montre un exemple de comparaison entre le langage UIML (cas B) et d'autres langages (cas A).

A. Pour définir un bouton, il y a plusieurs syntaxes selon la plate-forme utilisée :

Java : `JButton jbtood = new plastic-button ("Valider") .`
Html : `<input type="button" name=" plastic-button" value="Valider" />`

B. En revanche, **UIML** a une syntaxe unique pour toutes les plate-formes :

```
<part name="plastic-button" class="Button">
  <property name="content">Valider</property>
</part>
```

Et puis dans la partie **<peers>** :

- Pour la plate-forme Java :


```
<d-class name="Button" .. Maps-to="javax.swing.JButton" />
```
- Ou pour la plate-forme HTML :


```
<d-class name="Button" .. Maps-to="html:input"/>
```

Figure II.4 : Exemple de comparaison entre UIML et d'autres langages

Ce langage n'a aucun support explicite pour les modèles de tâche, les modèles de domaine, les modèles de dialogue ou encore ceux de contexte (malgré la notion de multiplateforme d'UIML). De plus, à chaque nouvelle plate-forme, les transformations des interacteurs abstraits AIO en interacteurs concrets CIO seront spécifiées, à nouveau, dans la partie *peers* du document UIML. Pour résoudre cet inconvénient d'UIML, un système de gestion de la présentation, ayant fait l'objet de travaux de thèse de [Ali *et al.*, 2004], peuvent être adoptés afin d'éviter la re-spécification des AIO à chaque plate-forme.

UIML est supporté par l'outil Liquid UI permettant d'éditer et de générer l'IHM. Cet outil inclut des moteurs de rendu (HTML, Java, J2EE (Java 2 Entreprise Edition), .NET (cadre de composants d'IHM destiné à Windows, Windows mobile et au Web), Symbian (Système d'exploitation de la nouvelle génération des téléphones portables intelligents), QT (Format des Widgets), VoiceXML, WML, CORBA pour la connexion à une base de données externe, Visual Basic, et C++).

III.2 AUIML

AUIML (*Abstract User Interface Mark-up Language*) est un langage prenant en compte l'intention (but) et les besoins de l'utilisateur. AUIML a été proposé par IBM [Clark, 2000; Merrick, 2001]. Il fait l'objet de la génération automatique d'IHM multiplateformes (Dynamic HTML, Java Swing, Palm-Pilot). AUIML est utilisé pour décrire l'interface en termes :

- d'éléments manipulés (données initiales, structures et données complexes telles que des images ou des sons) ;
- d'éléments d'interaction : types simples mêlant de la structure et des fonctionnalités (choix, groupe, table, arbre) ;
- et d'actions : permet de décrire un micro-dialogue pour la gestion d'évènements entre l'interface et les données.

AUIML dispose d'un mécanisme de formatage basé sur les grilles : chaque objet d'interaction abstrait alloue une cellule dans une grille prédéfinie. Il utilise également un gestionnaire de navigation qui aide à la construction de la structure de l'interface, et détermine le choix de navigation en fonction de la structure des données décrites. Une description AUIML intègre l'intention de l'utilisateur au niveau abstrait ; par exemple, pour le besoin de choix parmi un ensemble de possibilités, le concepteur ne décrit que des composants de présentation pour cette intention de l'utilisateur (boutons radio ou menu déroulant), si bien qu'AUIML associe à cette intention un ou plusieurs composants de présentation au moment de la distribution. Cette notion d'objectif ou d'intention se rapproche de la notion de tâche. Mais nous ne pouvons pas considérer que le langage AUIML supporte un modèle de tâche car il ne dispose pas de moyen pour exprimer l'enchaînement de tâches.

La Figure II.5 illustre l'exemple d'un document AUIML avec les objets d'interaction abstraits de l'IHM (en partie haute), et deux versions finales différentes de l'IHM. Au début, le concepteur choisit un composant abstrait *contrôle* qui sera affiché par la suite selon les besoins (quatre boutons sur une plate-forme Java, ou deux boutons dans une page Web).

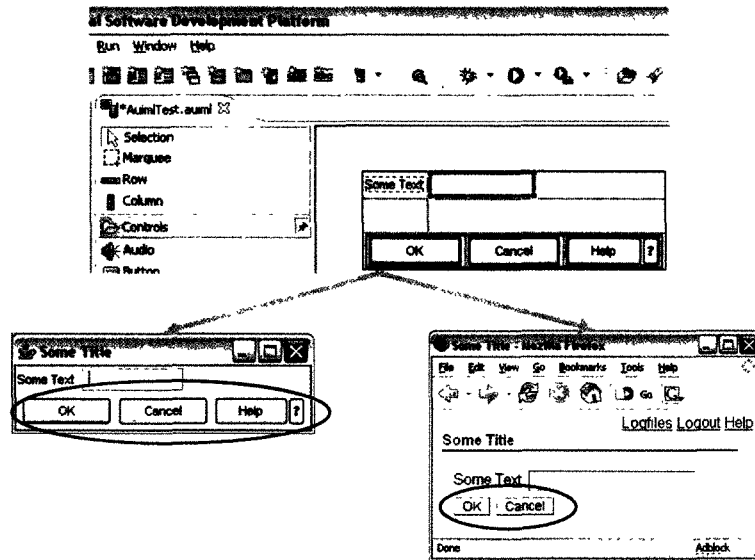


Figure II.5 : Environnement de développement AUIML avec un exemple d'une IHM associée à deux IHM finales différentes [www.ibm.com/developerworks]

AUIML résout l'un des principaux problèmes d'UIML puisque la description d'IHM est totalement indépendante de la plate-forme cible. Un autre point intéressant d'AUIML, par rapport à UIML est l'utilisation explicite d'un gestionnaire de navigation [Clark, 2000]. Ce gestionnaire construit la structure de l'interface, détermine le choix des systèmes de navigation en fonction de la structure des données décrites.

AUIML est supporté actuellement par un moteur de rendu Java et un autre pour le Web. Il ne supporte que les plates-formes Java Swing et HTML. Il existe un outil dit AUIML-VisualBuilder, distribué spécialement pour l'environnement de développement Eclipse

d'IBM⁶. AUIML-VisualBuilder permet aux concepteurs d'établir facilement des interfaces. Ensuite les interfaces spécifiées peuvent être déployée sur une plate-forme Java ou Web sans nécessité de changement du code.

III.3 XIIML

XIIML (eXtensible Interface Markup Language) [Puerta et Eisenstein, 2002] est un langage basé sur XML pour la représentation d'interfaces multiplateformes. Le langage XIIML peut être considéré comme une approche à base de modèles car il possède des modèles qui couvrent la plupart du cycle de vie d'une interface.

XIIML définit l'interface (cf. Figure II.6) à un niveau plutôt abstrait, comme un ensemble de composants. Ces composants peuvent spécialiser les tâches (processus métier), le domaine (définit une hiérarchie des composants) et l'utilisateur (définit une hiérarchie des utilisateurs prévus pour l'IHM). Ensuite les aspects concrets des interfaces sont définis grâce à d'autres composants qui décrivent l'espace de présentation, et le dialogue qui définit des actions à travers l'interface. Les composants sont ensuite reliés et l'IHM est générée et distribuée. Avec les précédents composants, quatre autres objets sont utilisés pour décrire l'interface. Ce sont les relations qui décrivent les liens entre les composants, et les attributs, à partir d'eux, il est possible de décrire les caractéristiques ou les propriétés d'un composant. XIIML fournit une séparation stricte entre la définition de l'interface et sa visualisation.

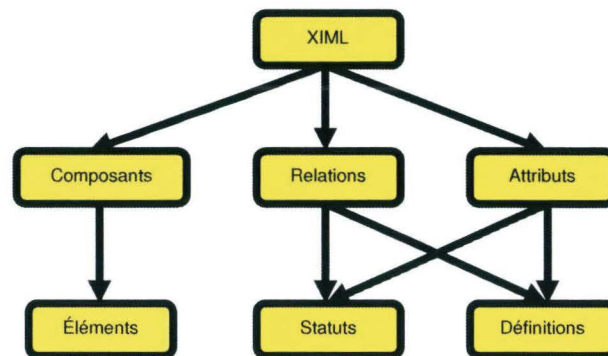


Figure II.6 : Structure représentative de base de XIIML

XIIML possède un moteur de rendu *Seguia* (System Export for Generating a User Interface Automatically) [Vanderdonckt, 1999a] permettant de générer une version d'IHM pour Windows, et un autre moteur *UI generator* qui génère des interfaces en WML (Wireless Markup Language : langage à balises utilisé par les téléphones portables). Enfin l'outil

⁶ Eclipse : est un environnement de développement intégré libre extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. La spécificité d'Eclipse IDE vient du fait de son architecture totalement développée autour de la notion de plug-in (les développeurs peuvent spécifier des outils supplémentaires à importer dans Eclipse afin de bien adapter leur environnement de développement avec Eclipse).

Vaquista (reVerse engineering of Applications through QUEStions, Information selection, and Transformation Alternatives) [Vanderdonckt *et al.*, 2001] permet de construire les modèles d'interface et de dialogue à partir d'une page Web.

III.4 XForms

XForms (XML Forms) [Dubinko *et al.*, 2000] est un langage à balises qui représente la nouvelle génération de formulaires pour le Web indépendamment de la plate-forme cible avec un niveau de description des interfaces utilisateur plutôt abstrait. Les formulaires XForms peuvent être utilisés avec XHTML (successeur de HTML v4.0 qui est supporté par la plupart des navigateurs Web actuels), ou WML. L'un des aspects essentiels de ce langage est la séparation des données, de la présentation et du contrôle. Cette séparation facilite la modélisation des données sans se préoccuper de savoir comment elles seront utilisées ou présentées. D'ailleurs les formulaires peuvent être réutilisés indépendamment de la page qui les contient. De plus le formulaire peut être présenté en plusieurs langues ce qui rejoint le principe d'internationalisation des IHM.

Un formulaire décrit avec XForms peut être découpé en trois parties (cf . Figure II.7) :

- Le modèle d'information se compose de deux parties : (1) la structure de donnée fournissant un schéma pour la description de l'instance des données, et les types des données ; (2) les extensions qui incluent des aspects qui ne sont pas typiquement exprimés dans le schéma de données, par exemple, des contraintes sur les entrées de l'utilisateur, ou des informations servant pour l'affichage ;
- Les données d'instance : à l'exécution, les entrées de l'utilisateur sont modélisées sous une forme XML afin de faciliter leur sauvegarde et de pouvoir les récupérer sur une nouvelle plate-forme ;
- L'interface home-machine est construite à base des composants déjà définis comme des balises, par exemple la balise `<xform:select>` équivalente à un composant de présentation permettant à l'utilisateur d'effectuer un choix parmi une liste d'items. Ensuite un moteur de rendu décore les composants de présentation, dans la page Web, adéquats à la plate-forme cible grâce à la technique CSS (Cascading Style Sheets). Par exemple la balise `<xform:select>` sera affichée sous forme de liste déroulante, ou boutons radio, etc., selon la capacité graphique ou encore de traitement de la plate-forme.

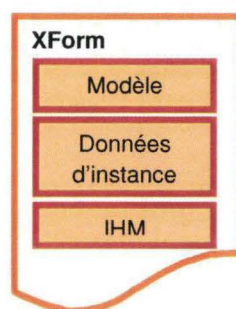


Figure II.7 : Structure conceptuelle de XForms

XForms offre aujourd'hui un degré d'abstraction suffisant, par rapport aux précédents langages comme UIML, pour la description des IHM indépendamment de la plate-forme. Comme les éléments de l'IHM sont définis de manière abstraite (en fonction des caractéristiques du support cible), le formulaire est présenté sans nécessité d'intervention de la part du concepteur ou de l'utilisateur. Cependant, XForms se limite à la modélisation des formulaires, et il est compatible, pour l'instant, uniquement avec les navigateurs Web supportant XML.

III.5 Plastic ML

Plastic ML [Rouillard, 2003] est un langage basé sur XML permettant de décrire des interfaces homme-machine de manière abstraite, donc indépendantes du support cible. Ce langage décrit des éléments d'entrée et de sortie d'une interface, à un haut niveau, autrement dit sans faire référence au périphérique qui sera utilisé lors de l'interaction. Grâce à des transformations XSLT (eXtensible Stylesheet Transformation), les documents en Plastic ML sont traduits automatiquement dans un langage à balises cible pour une utilisation concrète (HTML, WML, VoiceXML). Plastic ML est accompagné de son outil de développement Plastic ML Toolkit (cf. Figure II.8). C'est une boîte à outils qui sert à créer, modifier, et sauvegarder du code Plastic ML par manipulation graphique. L'un des avantages de Plastic ML est la gestion de la notion de rôle. L'utilisateur acquiert un certain rôle en se connectant à une base de données. Par la suite, l'interface affiche seulement à l'écran ce que cet utilisateur est autorisé à voir par rapport à son rôle.

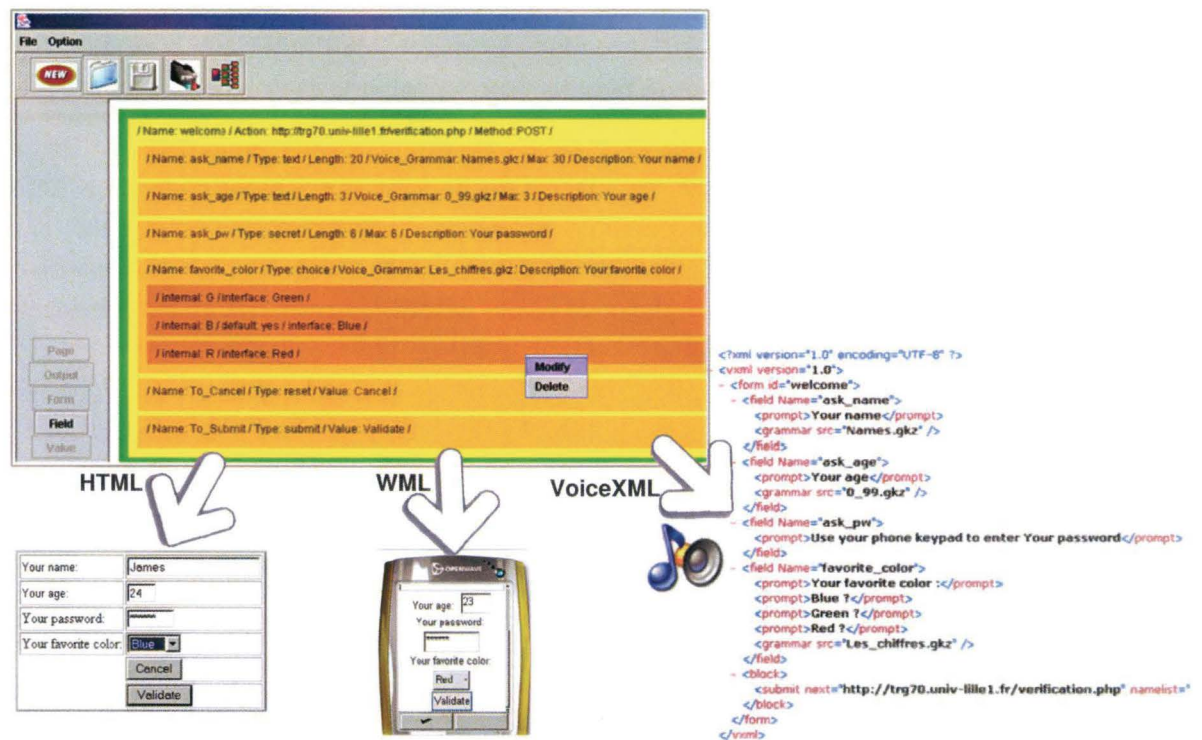


Figure II.8 : Toolkit de Plastic ML

Plastic ML est un langage en cours d'évolution, des nouvelles balises et des composants devraient être ajoutés. Il gère principalement les documents de style formulaires. Le nombre de balises prises en compte par Plastic ML est limité. Malheureusement les fonctionnalités plus riches (Java Swing ou AWT) ne sont pas couvertes par ce langage.

III.6 Projet Rainbow

Rainbow [Cheng *et al.*, 2004] est un cadre dédié à la conception de systèmes auto-adaptables. Il respecte la boucle de commande introduite par IBM pour la construction de systèmes autonomes. Rainbow utilise un modèle de composants hiérarchiques pour la représentation explicite de l'architecture du système. Le point fort de ce modèle est la possibilité de décrire les contraintes à vérifier régulièrement par le système d'administration avant et après la mise en place d'une politique d'adaptation. Les capteurs envoient des informations contextuelles au noyau fonctionnel qui se base, d'une part sur ces informations, et d'autre part sur l'architecture du système, pour décider du type d'adaptation et de sa mise en œuvre. Rainbow se base sur un modèle abstrait d'architecture pour superviser le système à l'exécution. Ce modèle représente les différentes parties du système sous une forme de composants. Il s'agit d'un modèle hiérarchique qui représente le système à des niveaux d'abstraction différents. Les liaisons entre les composants sont représentées sous forme de connecteurs. Les liaisons permettent d'exposer des interfaces internes au niveau d'un composant composite. Un système intégrant Rainbow expose une architecture en plusieurs couches (cf. Figure II.9) :

- La couche d'architecture : des jauges [Garlan *et al.*, 2001] permettent d'agréger les informations envoyées par les capteurs et de mettre à jour les propriétés appropriées dans le modèle d'architecture. Un gestionnaire de modèle traite les informations et accède au modèle d'information. Un évaluateur de contraintes vérifie périodiquement le modèle et détecte le besoin d'une adaptation si une contrainte est violée. Une machine d'adaptation détermine l'action à effectuer et met à jour l'adaptation correspondante.
- La couche de traduction : elle établit la correspondance entre le modèle et le système d'administration. Un dépôt dans l'infrastructure maintient la liste de correspondance entre la représentation en composants et l'implantation système. Par exemple, le système peut faire correspondre l'identificateur d'un composant à une adresse IP.
- La couche système : à ce niveau, des interfaces sont implantées. Des capteurs observent et mesurent les différents états du système. De plus, un service de découverte peut être sollicité pour allouer des ressources suivant des critères bien définis. Enfin, des actionneurs permettent la mise en place des modifications.

En se basant sur ces infrastructures, ajouter les politiques d'adaptation nécessite une connaissance spécifique du système tels que la plate-forme cible sur laquelle le système sera déployé, les paramètres des composants, leurs types, leurs propriétés et les contraintes de leurs comportements ainsi que les stratégies d'adaptation. Suivant la politique

d'administration et le contexte d'usage, Rainbow définit un style d'architecture. Le style d'architecture identifie les propriétés communes à un ensemble de systèmes. Afin de définir les politiques d'adaptation, Rainbow définit :

- les opérateurs d'adaptation qui correspondent à des opérations de base comme ajouter un service ou en éliminer un autre,
- les stratégies d'adaptation qui correspondent à des conditions dans lesquelles il faut effectuer les opérations d'adaptation.

Néanmoins, Rainbow présente quelques inconvénients. En effet, le but de ce projet est de se focaliser sur l'automatisation de l'adaptation mais le modèle de composants n'est exploité que pour la représentation du système adaptable et non pas pour la génération et la construction du système.

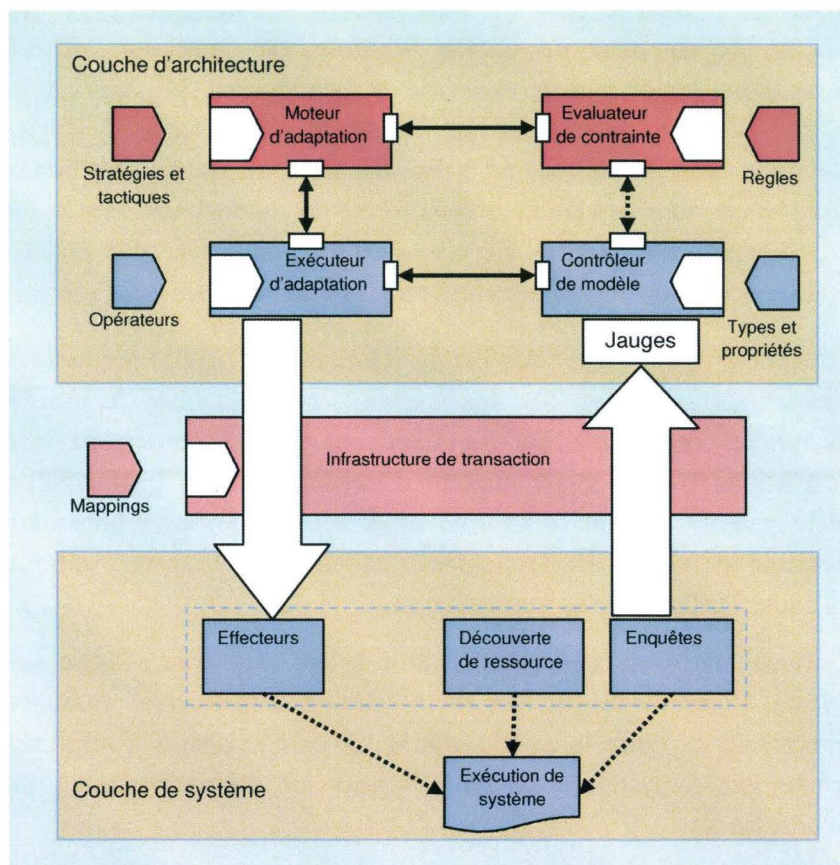


Figure II.9 : Cadre général du projet Rainbow [Cheng et al., 2004]

Plusieurs sous-projets sont développés dans le cadre du projet Rainbow comme *Noah*. C'est le premier prototype *Noah* permettant l'assemblage dynamique de composants

métier et techniques⁷. *Noah* permet de relier entre eux des composants Java, JavaRMI, EJB Jonas et .Net. Ce prototype, disponible depuis le site Web du projet Rainbow⁸, permet en particulier de faire évoluer dynamiquement les fonctionnalités d'un composant. Ensuite le projet ASPECT est apparu dans lequel le prototype AMUSING est conçu pour faciliter le développement par composition d'IHM indépendantes de la plate-forme. Ce prototype est détaillé ci-dessous.

III.7 AMUSING

AMUSING (Adaptable and Mergeable User Interface) est un environnement de développement conçu dans le cadre du projet Rainbow pour faciliter le développement par composition d'IHM indépendantes de la plate-forme. Il propose un modèle pour l'assemblage dynamique de composants métier, ceux technique et ceux d'interface, par la description du prototype AMUSING, dans le cadre du projet ASPECT. Ce prototype complète *Noah* [Blay-Fornarino *et al.*, 2002]. Il permet l'assemblage dynamique de composant d'IHM [Pinna-Dery *et al.*, 2003] et la liaison dynamique entre composants d'IHM et composants métiers en utilisant des interactions logicielles [Blay-Fornarino *et al.*, 2004].

AMUSING repose sur la définition d'une interface abstraite à l'aide du langage SunML (Simple Unified Natural Markup Language), puis sur la réification de cette description sous la forme d'un arbre abstrait permettant d'effectuer les différentes opérations comme la fusion des objets d'interface abstraits (ex. deux facettes de présentation) ; enfin une dernière étape consiste à projeter l'interface ainsi décrite vers une IHM concrète.

Dans le modèle de composants d'AMUSING, un composant métier peut être lié à un ou plusieurs composant(s) d'IHM à l'aide d'interactions logicielles (une interaction représente le contrôleur de communication). L'environnement de développement réalisé comprend un serveur d'interactions permettant d'enregistrer et d'utiliser des schémas d'interaction/coordination pour assembler des composants métiers et pour les faire interagir avec leurs composants techniques. Les schémas d'interaction sont spécifiés en ISL (Interaction Specification Language) [Pinna-Dery *et al.*, 2004]. ISL est un langage de spécification des interactions entre les composants. Lors de l'assemblage, le composant d'IHM doit être décrit de manière abstraite pour garantir l'indépendance de la plate-forme. Le langage SUNML est utilisé pour spécifier l'IHM indépendamment du support. Un composant d'IHM est décrit dans ce langage puis instancié sous forme d'arbre abstrait représentant la structure abstraite de l'IHM. Cette vue abstraite peut être ensuite projetée vers une ou plusieurs vues concrètes à l'aide d'un moteur de rendu (actuellement Swing ou Vocal). La vue abstraite et la ou les vues concrètes sont liées entre elles. La vue

⁷ Les composants techniques offrent des services spécifiques liés à la plate-forme comme la gestion des événements, la sécurité, etc.

⁸ <http://rainbow.i3s.unice.fr>

abstraite est utilisée comme un méta objet représentant à la fois la structure abstraite et conservant les données des IHM concrètes (valeurs saisies par l'utilisateur) qui doivent être synchronisées en permanence avec cette vue abstraite. Cette indirection supplémentaire apporte la flexibilité utile à la composition des IHM ainsi qu'à leur mobilité. La vue abstraite est liée au composant métier. Il est possible aussi de lier directement le composant métier à la vue concrète. Un tel lien direct peut être utilisé pour optimiser l'exécution lorsqu'un assemblage est figé lors de la conception. Mais dans ce cas, adaptation et assemblage dynamiques ne sont plus possibles.

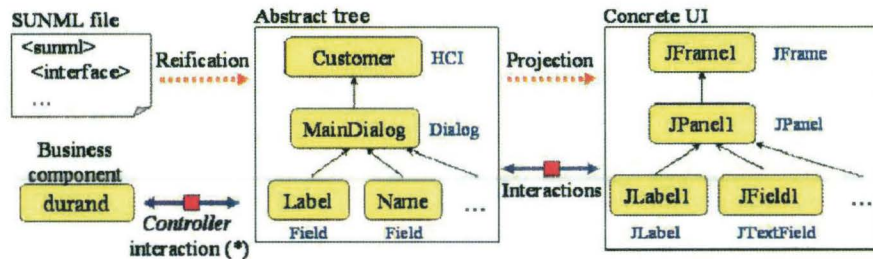


Figure II.10 : Architecture d'AMUSINGs

L'outil AMUSING permet d'éditer selon l'approche préconisée des interfaces utilisateurs en quatre vues différentes :

- 1) la première permet l'édition de la vue abstraite de l'interface dans le langage SunML ; cette description est ensuite réifiée en un arbre abstrait permettant sa manipulation et peut être projetée vers différentes interfaces concrètes. Aujourd'hui ces interfaces concrètes sont en Java Swing et VXML.
- 2) La seconde vue permet de fusionner manuellement selon différentes opérations des interfaces abstraites précédemment construites. Le résultat de cette fusion est une nouvelle interface abstraite qui peut à son tour être fusionnée ou projetée.
- 3) La troisième vue permet d'utiliser un langage de description des opérations de fusion (WML) afin d'automatiser ce processus.
- 4) La quatrième vue permet d'éditer et de visualiser les interactions logicielles entre les composants d'interface précédemment construits et les composants métier qui les utilisent.

De plus des algorithmes de fusion sont développés dans le cadre de ce prototype afin de permettre l'ajout statique et dynamique de composants d'IHM à une IHM existante.

III.8 MOBI-D

MOBI-D (MOdel-Based Interface Designer) [Eisenstein *et al.*, 2001] est un environnement de conception qui permet de réaliser des conceptions orientées utilisateurs grâce à une méthodologie de développement d'interface multicible à base de modèle. Cette méthode permet d'effectuer une adaptation de la présentation en fonction de la plateforme cible. MOBI-D s'appuie sur cinq composants (cf. Figure II.11) :

- **Un modèle de l'utilisateur** définit les types des utilisateurs et leurs particularités.
- **Un modèle de la tâche utilisateur** propose une représentation des tâches que l'utilisateur peut réaliser à travers l'IHM.
- **Un modèle du domaine** définit les objets auxquels l'utilisateur accède via l'interface. Les objets du domaine sont organisés en classes et propriétés associées.
- **Un modèle de présentation** est une vue des caractéristiques statiques de l'interface. Ce modèle est généré à partir du modèle du domaine. Le choix des interacteurs est réalisé par un outil de conception automatisée nommé TIMM (*The Interface Model Mapper*) [Eisenstein et Puerta, 2000] en s'appuyant sur un arbre de décisions. Cet arbre permet de sélectionner des objets interactifs abstraits (AIO) en fonction du concept du domaine à représenter (ex : un AIO de choix est spécifié simplement comme *champ de choix*). Ensuite les objets d'interaction abstraits sont concrétisés en interacteurs physiques (CIO) en fonction de la plate-forme (ex. en Swing⁹ : *champ de choix* => *Jlistbox*).
- **Un modèle de dialogue** spécifie les commandes utilisateurs, les techniques d'interaction, les réponses de l'interface et les séquences de commandes que l'interface permet pendant une interaction avec l'utilisateur. Ce modèle est généré automatiquement et dérivé particulièrement à partir du modèle de la tâche utilisateur.

MOBI-D sert essentiellement à effectuer une conception d'interface structurée et automatisée à partir de modèles abstraits (modèle de domaine, modèle de tâche) pour obtenir des modèles concrets (modèle de présentation, modèle de dialogue). Afin de permettre une génération semi-automatique de l'interface, nous distinguons deux grandes familles de composants au sein des divers modèles :

- Les composants abstraits qui donnent une vue abstraite des tâches et du domaine, et qui ne peuvent pas être manipulés directement à travers l'interface.
- Les composants concrets qui se trouvent dans les modèles de dialogue et de présentation ; les composants de ce type peuvent être manipulés directement au travers de l'interface (fenêtres, boutons, clic souris...).

MOBI-D contient une base de connaissances (guides de style, patrons de conception...) à propos de directives et de conseils opérationnels pour la conception d'interface afin d'automatiser la conception des interfaces et de venir en aide au concepteur. MOBI-D intègre un certain nombre d'outils de conception et d'implémentation incluant :

⁹ Swing est une bibliothèque graphique intégrée dans J2SE (Java 2 Standard Edition). Swing utilise le principe du modèle MVC (cf. chapitre I, §II.2.1) et dispose de plusieurs choix d'apparence pour chacun des composants standards.

- U-TEL (User-Task Elicitation Tool) [Tam *et al.*, 1998], outil de génération du modèle de tâche de l'utilisateur directement à partir des experts du domaine ;
- Un ensemble d'éditeurs de modèles. Chaque type de modèle est manipulé par un éditeur interactif spécifique ;
- TIMM (Task-Interface Model Mapping tool) [Eisentien et Puerta, 1998] est un outil pour faire les liens entre les modèles ;
- Enfin le constructeur d'IHM (Interface Builder).

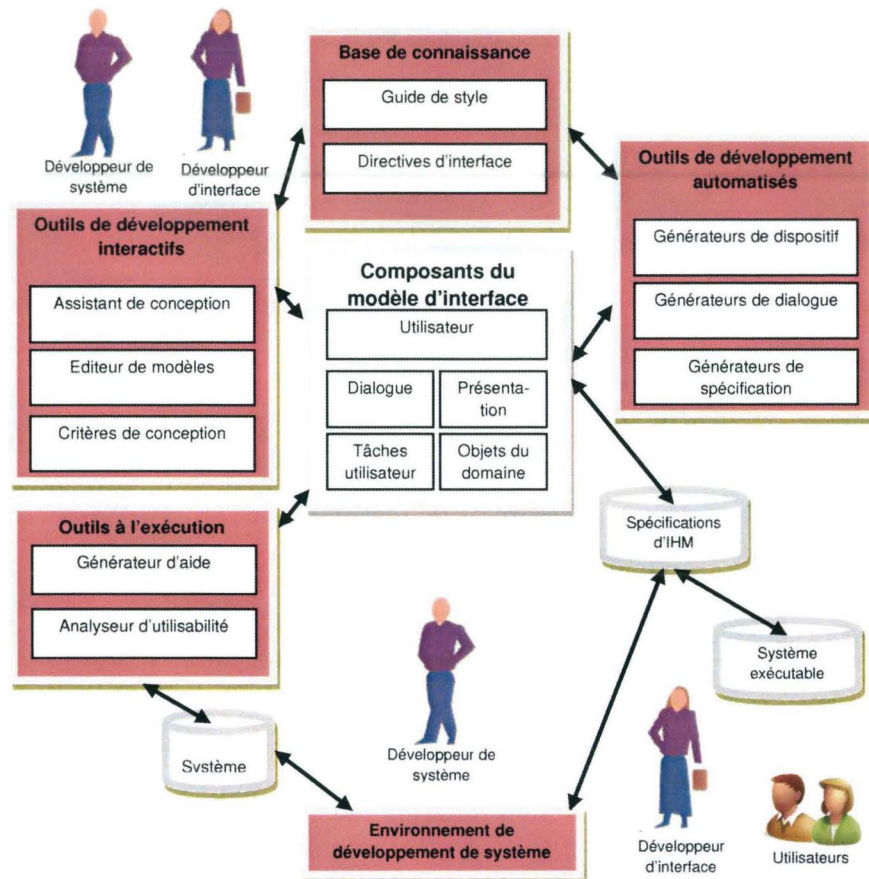


Figure II.11 : Architecture de MOBI-D [Puerta et Eisenstein, 1998]

Du point de vue des IHM multicibles, MOBI-D se fonde sur une modélisation des plateformes cibles, du modèle de tâche et de différents modèles de présentation. L'interface graphique est générée en sélectionnant des interacteurs et une structure de présentation parmi un ensemble de structures de présentation prédéfinies. Donc l'adaptation est totalement statique et prédéfinie par le concepteur du système. Après la distribution du système vers la disposition cible, aucune adaptation ne sera réalisée.

III.9 Projet Cameleon

Le projet Cameleon (Context Aware Modelling for Enabling and Leveraging Effective interactiON) [Calvary *et al.*, 2003] propose un cadre de référence qui unifie les approches

à base de modèles ayant la modélisation des tâches comme point de départ pour la conception d'IHM multiplateformes. Il structure le processus de conception et génération d'IHM en quatre niveaux de réification ou concrétisation. La Figure II.12 illustre une version simplifiée du cadre Cameleon comprenant quatre niveaux :

1. tâches et concepts : dans cette étape, des spécifications du système sont décrites en termes de tâches utilisateur, et d'objets du domaine qui seront manipulés par les tâches.
2. Interface abstraite : l'espace d'interaction est défini en groupant les tâches selon différents critères "indépendamment" de la plate-forme cible et de la modalité ;
3. Interface concrète : l'interface abstraite est concrétisée pour un contexte d'usage (une modalité mais plusieurs contextes d'usage) afin de définir les objets concrets de l'interaction qui sont employés plus tard pour décrire la version finale de l'interface ;
4. Interface finale : c'est la version opérationnelle de l'interface, autrement dit la version finale qui peut être déployée sur la plate-forme cible.

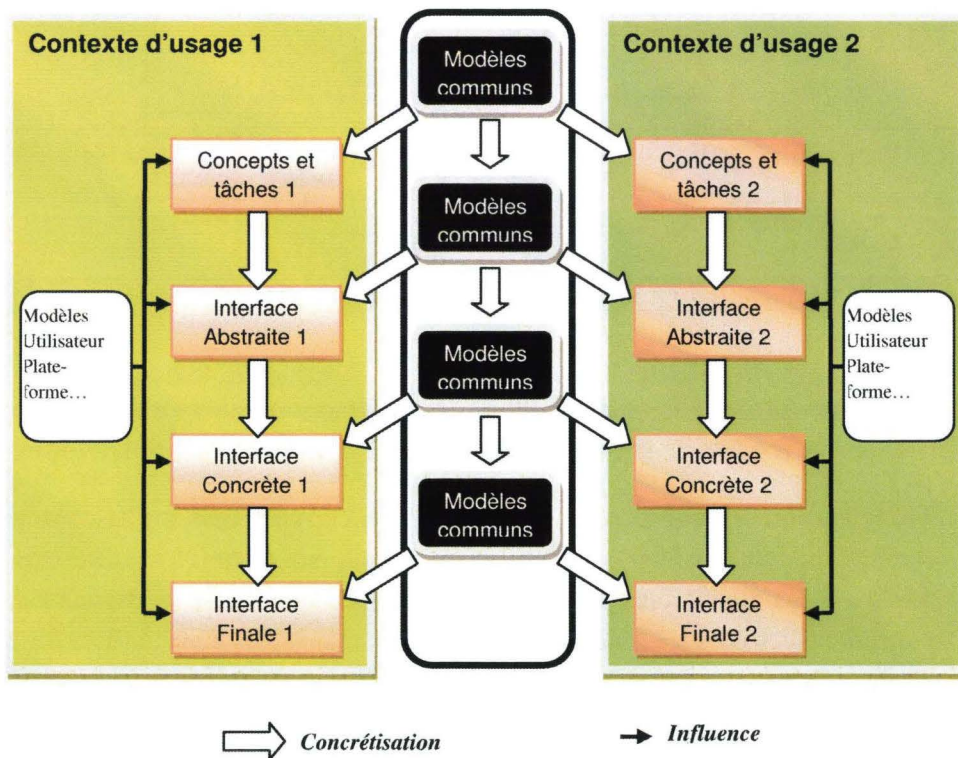


Figure II.12 : Approche à base de modèles pour l'adaptation des interfaces (adaptée de [Calvary et al., 2003] par [Tarpin, 2006])

En se référant au cadre proposé lors du projet Cameleon, des outils et méthodes ont été développés ; parmi eux, nous présenterons par la suite Teresa et UsiXML.

III.9.1 Teresa

L'approche Teresa (*Transformation Environment for InteRactive Systems representAtions*) [Mori *et al.*, 2004] est développée dans le cadre du projet Cameleon. Elle s'inscrit dans la conception à base de modèle (cf. Figure II.13).



Figure II.13 : Vue générale de l'environnement de Teresa

Teresa se base sur un ensemble de modèles et d'outils pour construire une interface adaptée à la plate-forme. Elle se compose de quatre étapes permettant aux concepteurs de commencer par un modèle de tâche, puis de générer l'IHM finale multiplateforme :

- **Modélisation à haut niveau des tâches de l'application multi-contexte.** Dans cette étape, un premier modèle est construit, contenant les contextes d'usage prévus, et les divers rôles impliqués ; en parallèle un modèle de domaine est généré pour identifier tous les objets qui doivent être manipulés pour réaliser les tâches du système et les liens entre les objets. Ces modèles sont construits par l'intermédiaire de l'outil CTTE [Mori *et al.*, 2002].
- **Développement d'un modèle de tâche pour les différentes plates-formes considérées.** Ici les concepteurs doivent filtrer le modèle de tâche selon la plate-forme cible afin d'avoir un modèle de tâche pour la plate-forme considérée. Dans cette étape, pour chaque tâche le concepteur peut spécifier la plate-forme (PC, PDA, téléphone portable) sur laquelle cette tâche peut être exécutée.
- **Du modèle de tâche à l'interface abstraite.** Une description abstraite de l'interface est obtenue. Elle est composée d'un ensemble de présentations qui sont

identifiées par une analyse des liens entre les tâches. Chaque présentation est structurée au moyen d'un interacteur composé de divers opérateurs.

- **Génération d'interface.** Cette phase dépend complètement de la plate-forme cible. L'IHM est générée en prenant en compte les propriétés spécifiques du dispositif cible. En effet, les éléments de l'interface concrète doivent satisfaire les exigences de l'affichage, des composants de présentation, du système d'exploitation et de la boîte à outils.

L'approche TERESA se base sur la modélisation de tâches et plus concrètement sur les relations temporelles et logiques du modèle de tâche comme modèle du dialogue. La liaison avec le noyau fonctionnel se fait manuellement. Il est cependant difficile de savoir clairement comment la description de l'interface est connectée avec les objets de l'application.

III.9.2 UsiXML

UsiXML (*USer Interface eXtensible Markup Language*) [Limbourg *et al.*, 2004] est un langage utilisé dans une méthode de spécification et de génération d'IHM plastique basée sur XML. C'est une démarche complète avec des outils et des environnements de développement pour chaque niveau de génération d'IHM plastique. Contrairement à la plupart des langages basés XML, UsiXML supporte différents modèles pendant le processus de génération d'IHM plastique (modèles de tâche, d'interface abstraite, d'interface concrète, d'interface finale, de contexte, de transformation, et enfin de correspondance). Chaque modèle défini dans UsiXML peut être utilisé indépendamment des autres modèles. De plus, pour chaque modèle, des outils sont disponibles ; ils sont positionnés dans la Figure II.14.

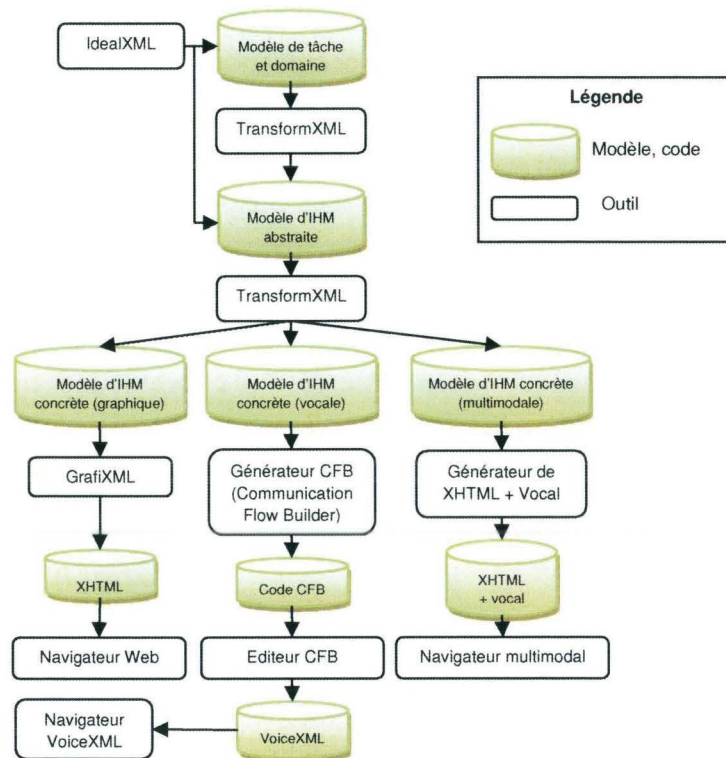


Figure II.14 : Modèles et outils d'UsiXML [Stanciulescu et al., 2005]

UsiXML possède aussi un modèle de présentation ; les transformations entre les modèles sont appliquées principalement à ce niveau. Ce modèle supporte la transformation des objets d'interaction pour lesquels il existe une description de cette transformation dans le modèle de transformation UsiXML (Figure II.15).

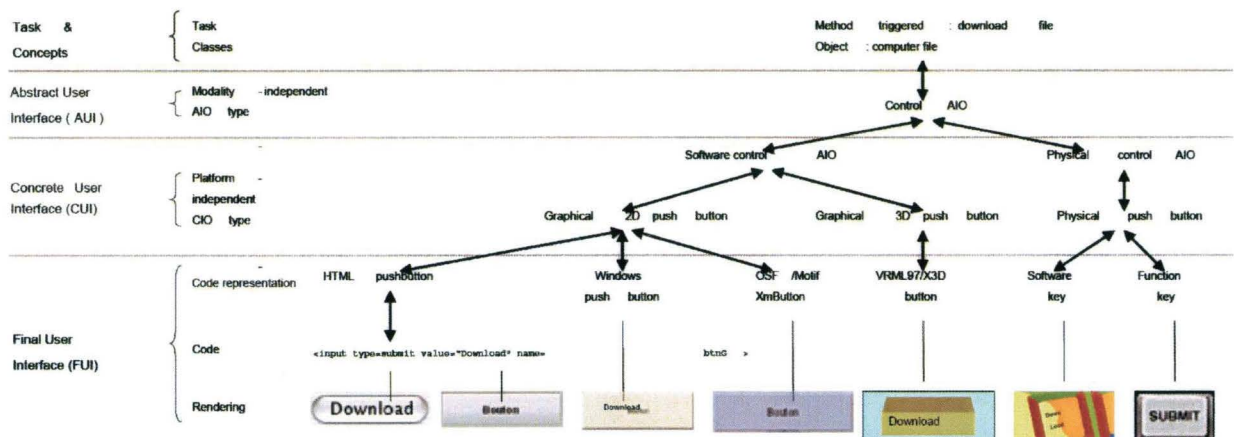


Figure II.15 : Exemple de transformations en UsiXML entre les niveaux d'abstraction de Cameleon (extrait de [Vanderdonckt et al., 2004])

Par exemple, IdealXML est un éditeur graphique pour la modélisation de tâches, de données et du modèle d'interface utilisateur abstrait. Cet outil gère aussi la création de modèle de correspondance entre ces trois premiers modèles. Il permet donc de transformer

un modèle vers l'autre. Par exemple, le modèle d'interface abstraite peut être généré à partir d'un modèle de tâche déjà spécifié (cf. Figure II.16).

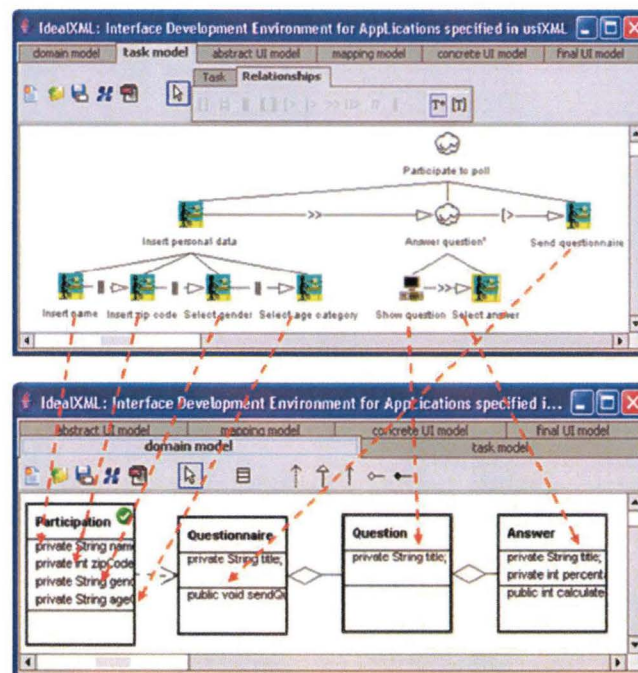


Figure II.16 : Transformation du modèle de tâche vers le modèle de domaine avec l'outil IdealXML [Stanciulescu et al., 2005]

L'outil TransformiXML [Stanciulescu et al., 2005] supporte des règles basées sur des transformations entre les différentes couches d'abstraction. Cet outil peut être utilisé pour générer le modèle d'interface abstraite ou aussi d'établir plusieurs versions du modèle d'interface concrète en fonction de la modalité (Web, Voice...). Pour les notions de réutilisation, Lepreux propose une base théorique pour fusionner statiquement les dispositions existantes de l'IHM dans le cadre de la dégradation de l'IHM pendant la migration vers une autre modalité [Lepreux et al., 2006b]. [Florins et al., 2006] ont introduit la notion de règles de dégradation élégante, en se basant sur une spécification multicouche dans un langage de description des IHM. Les règles ne sont appliquées qu'à la conception.

Des éditeurs graphiques supportent différents types de design comme *SketchiXML* et *GrafiXML*. *GrafiXML* est un outil graphique permettant de concevoir et manipuler des IHM ; il génère automatiquement la description correspondante à *UsiXML*. *SketchiXML* est un éditeur d'interface à niveau de fidélité basse (voir les quatre niveaux de fidélité [Vanderdonckt et Coyette, 2006], cf. Figure II.17). Il permet au concepteur ou à l'utilisateur final de concevoir une interface graphique indépendante de la plate-forme au niveau de fidélité basse. L'IHM dessinée est analysée afin de générer des spécifications de l'IHM désirée indépendamment de tout contexte d'usage. Ensuite les spécifications peuvent être importées par un éditeur d'interface à niveau de fidélité élevée comme *GraphiXML* afin de générer l'IHM finale (cf. Figure II.18). *SketchiXML* est supporté par

un système de reconnaissance de forme, un système de reconnaissance de geste, et un système de reconnaissance de l'écriture [Vanderdonckt *et al.*, 2006].

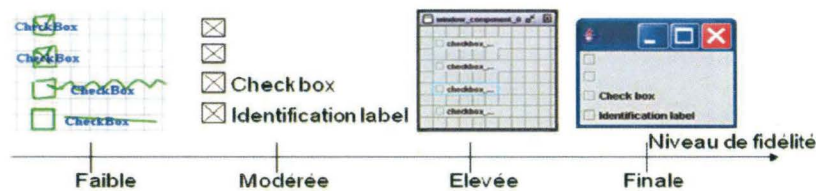


Figure II.17 : Niveaux de fidélité de l'IHM [Vanderdonckt et Coyette, 2006]

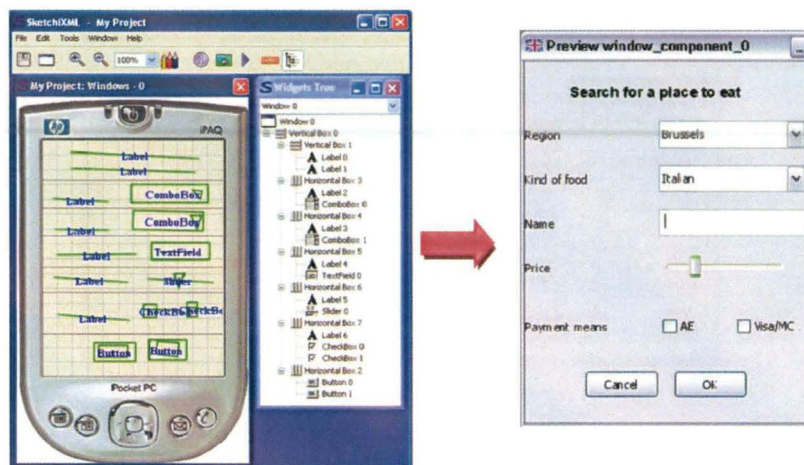


Figure II.18 : IHM finale générée à partir des spécifications à niveau de fidélité basse en SketchiXML [UsiXML.org]

Parmi les autres outils existants, *Vaquista* supporte le retour en arrière (notion de *rétro-ingénierie*) ; autrement dit il permet de remonter d'une page Web existante au niveau d'interface abstraite ou aussi celui d'interface concrète [Bouillon *et al.*, 2004]. Finalement UsiXML possède des moteurs de rendu comme *FlashiXML* et *QtXML* qui permettent de générer différents formats d'interfaces finales. La Figure II.14 illustre l'ensemble des modèles avec des outils qui peuvent être appliqués pendant la transformation d'un modèle à l'autre.

UsiXML est actuellement le langage de modélisation d'IHM le plus expressif et le plus riche en outils. Il supporte de nombreux modèles et est construit sur une base formelle. Cependant, la complexité relative des modèles et le nombre élevé d'outils associés nécessitent un effort considérable de la part du concepteur pour maîtriser complètement cette approche. De plus certains des outils sont en cours de proposition, d'amélioration ou de test.

Nous utiliserons dans notre démarche, le modèle d'interface abstraite d'UsiXML avec quelques points d'amélioration ou encore d'évolution ; par exemple le modèle de tâche sera intégré dans le modèle d'interface abstraite afin de retourner à la hiérarchie de tâches

à l'exécution pendant le processus d'adaptation. C'est pourquoi le modèle d'interface abstraite d'UsiXML sera détaillé dans le chapitre III.

III.10 L'approche par *Comet*

Cette approche est le résultat d'une fusion, avec une visée de plasticité d'IHM, entre une approche à base d'interacteurs adaptatifs [Crease, 2001] et une démarche à base de modèles pour l'adaptation des systèmes interactifs [Calvary *et al.*, 2005]. Elle adopte une nouvelle génération d'interacteurs appelés *Comets* (COntext Modable widgETs), pour la construction de systèmes interactifs plastiques. La *Comet* est définie comme un interacteur auto-descriptif, c'est-à-dire un mini-système interactif, éventuellement doué de facultés d'adaptation. Une *Comet* est capable de s'adapter ou d'être adaptée au contexte d'usage, capable d'être recrutée (versus éliminée) lorsqu'elle est apte (versus inapte) à couvrir le nouveau contexte d'usage. Un interacteur peut être une entité logicielle et/ou matérielle dont l'usage suppose la mise à disposition de ressources de calcul, de communication et d'entrée/sortie pour la réalisation des actions composant les tâches. L'auto-description porte sur les propriétés fonctionnelles et non fonctionnelles de la *Comet*. Les propriétés fonctionnelles définissent les services rendus par la *Comet*, c'est-à-dire les tâches utilisateur qu'elle supporte et les concepts du domaine qu'elle manipule. Les propriétés non fonctionnelles qualifient le service rendu en précisant, par exemple, la performance ou la tolérance aux fautes [Chung, 1991 ; Rosa *et al.*, 2002]. Trois types de *Comet* peuvent être distingués : les *Comets* auto-descriptives, polymorphes et auto-adaptatives. Conformément aux autres approches à base de modèles, la *Comet* se définit selon quatre niveaux d'abstraction :

- concepts et tâches (dit « abstraction » de la *Comet*), qui exporte les concepts et tâches que la *Comet* représente ;
- l'interface abstraite, qui publie sa structuration en espaces de travail ;
- l'interface concrète, qui exporte son style (par exemple, le style bouton) et précise si ce style est typique dans le contexte d'usage considéré ;
- l'interface finale (i.e. la version exécutable ou interprétable de la *Comet*), qui gère la capture/réception/diffusion du contexte d'usage ainsi que l'état d'interaction. La *Comet* peut être polymorphe sur un ensemble de contextes donnés, auto-adaptative et/ou plastique pour un ensemble de propriétés données. Le polymorphisme peut s'exercer à différents niveaux d'abstraction (concepts et tâches, interfaces abstraite, concrète et finale).

La *Comet* peut en plus embarquer des modèles d'évolution et de transition pour diriger son adaptation. L'implémentation des *Comets* s'appuie sur le modèle Compact (cf. chapitre I, §II.2.2).

Si nous partageons l'avis sur l'apport bénéfique de l'intégration d'un support architectural dans une approche à base de modèles pour l'adaptation des systèmes interactifs, nous

pensons qu'encapsuler dans le même composant logiciel (la *Comet*) toutes les spécifications des modèles, ainsi que des mécanismes d'adaptation, peut surcharger le composant particulièrement dans des domaines d'application à forte densité d'interacteurs.

III.11 Approche de Thevenin

Thevenin a proposé un environnement de génération d'IHM plastique à base de modèles, appelé ARTStudio [Thevenin, 2001] (Adaptation par Réification et Traduction Studio). Dans sa thèse, Thevenin propose un cadre conceptuel et une démarche en plusieurs étapes pour chaque fois générer les modèles de l'étape suivante à l'aide de règles préétablies. Dans le cadre conceptuel, la description des modèles est augmentée par deux principes :

- Le principe de factorisation (cf. Cameleon, §III.9) qui explicite les descriptions communes à plusieurs contextes,
- et le principe de décoration (cf. Teresa, §III.9.1), dans lequel des descriptions, des factorisations et décorations s'ajoutent aux démarches de réification.

Pour générer une interface, ARTStudio s'appuie sur cinq modèles de description : le modèle des tâches, celui des concepts, celui des instances, celui des interacteurs et celui de la plate-forme cible.

- Le modèle des tâches décrit, sous la forme d'un graphe, les tâches qui seront réalisées au travers de l'IHM.
- Le modèle des concepts décrit la structure des données qui seront manipulées par l'IHM.
- Le modèle des instances décrit les instances de concepts manipulés par l'interface et le graphe de dépendance entre ces instances.
- Le modèle des interacteurs décrit les interacteurs qui sont présents sur la boîte à outils graphique de la plate-forme visée.
- Le dernier modèle décrit la plate-forme pour laquelle l'interface sera produite.

D'un point de vue multiplateformes, deux mécanismes sont implémentés (Figure II.19) :

- Une réification avec point de divergence entre les IHM abstraites et concrètes. C'est le mécanisme permettant la production d'interfaces graphiques finales pour les plateformes Mac/PC et Pilot (côté gauche de la Figure II.19) ;
- Une traduction par simple copie. C'est le mécanisme qui permet de créer une version propre aux plateformes WAP (côté droit de la Figure II.19).

Mais cette IHM n'est plastique qu'au regard de la plate-forme. En outre, l'utilisateur ne peut changer de plate-forme à la volée (l'IHM doit, à chaque fois, être renvoyée à l'étape de conception pour passer à une nouvelle plateforme). ARTStudio, dans sa version actuelle, n'a pas incorporé de modèle d'évolution. Ce modèle spécifie les réactions à mettre en œuvre en cas de changement de contexte. ARTStudio présente un support graphique pour les étapes de génération d'interface plastique. Cependant, ce support ne

couvre pas tous les principes du cadre conceptuel proposé dans la thèse [Thevenin, 2001]. Il se focalise sur la démarche de réification en occultant la démarche de traduction, et ne propose pas de support pour le principe de décoration. L'interface utilisateur générée par ARTStudio est une interface graphique en Java.

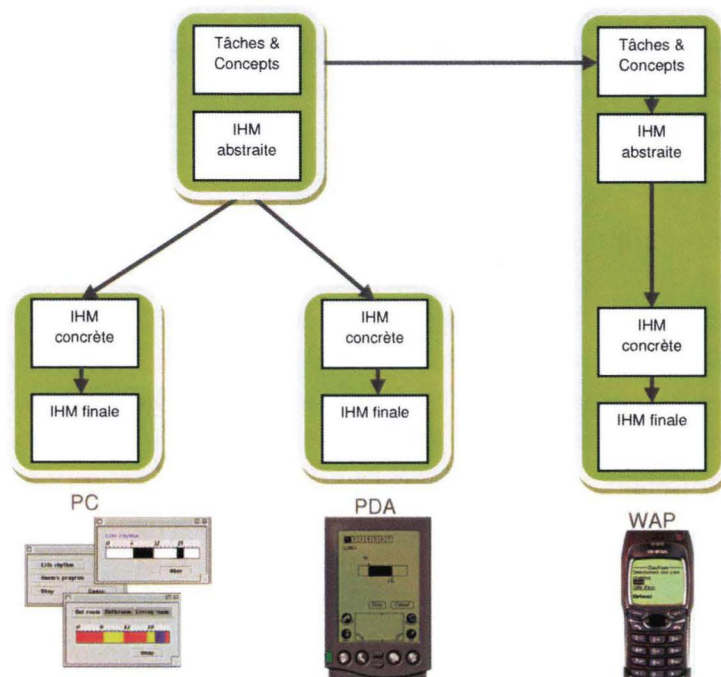


Figure II.19 : Exemple lié au processus de développement d'ARTStudio pour la production d'interfaces multiplateformes

III.12 Synthèse

Sans souci d'exhaustivité mais plutôt avec celui de représentativité, nous avons passé en revue un certain nombre de méthodes et de langages pour la spécification et la génération d'IHM multiplateforme et/ou orientée contexte. La plupart des langages de spécification sont des langages à balises. Dans un premier temps, nous avons présenté une première technique d'adaptation basée sur la spécification des présentations multiples (structures) parmi lesquelles une structure sera sélectionnée en fonction de la plate-forme cible. A côté de cette technique, les composants d'IHM ne sont définis qu'à la dernière partie du document XML pour laisser la possibilité au concepteur de spécifier les composants en fonction du type du dispositif cible. Cette solution est utilisée essentiellement dans le langage UIML. Par exemple une balise `<part class="Button" id="PlasticBouton" />` dans UIML sera traduite par le moteur de rendu Java Swing en `JButton` ou par une balise `<Button name="PlasticBouton"/>` par un moteur de rendu HTML.

L'idée d'UIML et d'XIML est de trouver un langage canonique pour spécifier les interfaces multicibles, sans nécessité de spécifier des versions différentes d'une IHM pour diverses plateformes. Cependant, XIML couvre bien la partie initiale du cycle de développement d'IHM (partie abstraite), alors qu'UIML se trouve dans la partie concrète du cycle de développement d'IHM. Donc la capacité d'UIML comme un métalangage

dans la phase de la concrétisation d'IHM et les composants abstraits de XIML, peuvent être mis en collaboration afin de trouver un langage standard pour spécifier les IHM multiplateforme en couvrant tout le cycle de vie d'IHM. A ce titre, notons des travaux de recherche visant à définir mieux les relations et la coopération entre UIML et XIML. D'ailleurs, XIML a été appliqué dans différentes approches, visant à exploiter son principe à base de modèle. Par exemple, le langage XIML est utilisé comme une technologie de base pour l'outil *DiaTask* d'extraction du modèle de navigation à partir d'un modèle de tâche [Forbrig *et al.*, 2004].

Nous avons envisagé des approches reposant sur une description des besoins de l'utilisateur ; ensuite les descriptions abstraites des besoins seront traduites juste avant l'implémentation en composants de présentation selon le support cible. Par exemple, les langages XForms et AUIML permettent d'exprimer que le besoin de l'utilisateur est d'effectuer un choix. Cette intention peut alors être concrétisée par un ensemble de boutons radio ou par un menu déroulant selon les caractéristiques capacitaires de la plateforme. Pour effectuer la description de l'interface abstraite et pour la concrétisation de cette interface, certaines approches s'appuient sur des outils. Plastic ML s'appuie aussi sur des descriptions abstraites de l'IHM. Ensuite les abstractions de l'IHM sont traduites automatiquement vers des langages idoines (XHTML, WML, ou VoiceXML). Chaque approche s'appuie sur des outils qui couvrent totalement ou partiellement l'approche.

Les Tableaux II.1 et II.2 présentent un récapitulatif des méthodes et langages étudiés en spécifiant leurs modèles, leur méthodologie employée, les langages cibles, les plateformes supportées, le type d'adaptation, leur capacité d'apprentissage, et les outils associés à chaque approche. Les approches à balises présentées n'ont généralement que deux modèles : un pour la présentation et un autre pour le dialogue à travers de l'IHM.

Nous avons constaté qu'une approche (XForm) s'intéresse seulement aux IHM basées sur les formulaires Web. Cela limite l'application de l'approche pour des systèmes interactifs plus complexes. Nous avons présenté la méthode MOBI-D comme un exemple d'approche de type MBD se basant sur un ensemble de modèles, et où la description de l'interface consiste à définir des objets d'interaction abstraite (AIO) ; ensuite les objets AIO seront remplacés par des objets d'interactions concrets (CIO) compatibles avec le support cible.

Nous décrivons dans les tableaux ci-dessous les approches orientées contexte. Ces approches sont à base de modèles, elles visent l'adaptation des systèmes interactifs au contexte d'usage. Ces approches se différencient par leurs capacités d'adapter l'IHM, le temps d'application de l'adaptation, l'évolution de l'adaptation, et sûrement la gamme du contexte d'usage prise en compte. Chaque approche possède un certain nombre de modèles qui couvrent le cycle de développement d'IHM plastique. Ces modèles peuvent être spécifiques à des sous parties du contexte. Par exemple le modèle d'interface concrète est construit à partir du modèle d'interface abstraite avec des spécifications concernant le contexte d'usage. Ainsi, nous constatons que certaines approches se concentrent sur le modèle de tâche (ex : Teresa) comme modèle central pour l'adaptation et d'autres mettent le modèle de tâche en collaboration avec d'autres modèles jusqu'à un certain niveau.

Pendant le cycle de développement de Teresa, les tâches doivent être filtrées selon la plate-forme cible afin d'avoir des tâches exécutables sur la plate-forme considérée.

D'ailleurs, certaines approches proposent un support pour chaque modèle (ex : UsiXML) accompagné d'une gamme d'outils pour relier ces modèles. UsiXML est très intéressant dans la mesure où il permet aux développeurs de systèmes sensibles au contexte, de s'intégrer à n'importe quelle étape de la méthode et d'utiliser les outils existants ou de développer des nouveaux outils. D'autres approches préfèrent encapsuler la description des modèles dans un composant (ex : *Comet*) en le dotant ainsi de capacités d'auto-description d'adaptation. Aujourd'hui, et dans un objectif de normalisation, des travaux considérables sont menés pour reproduire les approches à base de modèles sur l'architecture MDA (*Model Driven Architecture*) [Sottet *et al.*, 2005].

Cependant la possibilité d'évoluer et d'améliorer la qualité des réactions d'adaptation est absente dans les approches présentées ; de plus le changement contextuel oblige l'IHM à être transformée à la conception pour appliquer l'adaptation pré-calculée (statique), sauf la *Comet* qui permet de passer d'une présentation prédéfinie à l'autre, à l'exécution, sous réserve de trouver des descriptions de l'adaptation demandée pour le changement au contexte détecté (application dynamique de l'adaptation statique). Le Tableau II.2 présente un récapitulatif des différentes approches orientées contexte pour l'adaptation des systèmes interactifs introduits.

Tableau II.1 : Méthodes et langages pour la spécification et la génération d'IHM multiplateforme et/ou orientée contexte –Partie I

	1-Langage à balises	2-Méthode	3-Modèles	4-Méthodologie	5-Multiplateforme/ Multi-contexte (intégrant la plate-forme)
UIML	X		IHM abstraite, Dialogue, IHM finale.	Spécification de présentations d'IHM multiples avec des décorations correctives.	Plate-forme
AUIML	X		IHM abstraite, Dialogue, IHM finale.	Spécification d'une description générique des IHM totalement indépendante de la plate-forme cible, ensuite le moteur de rendu génère les IHM finales et fait la décoration.	Plate-forme
XIML	X		Tâches, Domaine, Dialogue, IHM abstraite, IHM concrète, IHM finale.	Collaboration entre différents composants associés aux niveaux abstrait et concret.	Plate-forme
XForms	X		Information, données, IHM abstraite, IHM finale.	Spécification abstraite des formulaires Web, modélisation des entrées de l'utilisateur en XML.	Plate-forme
Plastic ML	X		IHM abstraite, Dialogue, IHM finale.	Spécification d'une description générique des IHM.	Plate-forme
AMUSINg	X	X	Domaine, Tâches, IHM abstraite, IHM concrète, IHM finale.	Assemblage dynamique de composants métier, technique et de présentation.	Plate-forme
MOBI-D		X	Tâches, Domaine, Utilisateur, Dialogue, IHM abstraite, IHM concrète, IHM finale.	structures de présentation prédéfinies.	Plate-forme
Teresa		X	Domaine, Tâches, IHM abstraite, IHM concrète, IHM finale.	Centrée tâche, génération associée.	Plate-forme
UsiXML	X	X	Domaine, Tâches, IHM abstraite, IHM concrète, IHM finale, Contexte, Transformation, Correspondance	Collaboration entre modèles, description générique, génération associée.	Contexte (Plate-forme, Utilisateur (limité), Environnement (limité))
Comet		X	Domaine, Tâches, IHM abstraite, IHM concrète, IHM finale, Transition, Evolution.	Couplage interacteur et approches à base de modèles.	Contexte (Plate-forme, Utilisateur, Environnement)
Approche de Thevenin		X	Domaine, Tâches, IHM abstraite, IHM concrète, IHM finale.	Centrée tâche, Génération assistée.	Plate-forme

Tableau II.2 : Méthodes et langages pour la spécification et la génération d'IHM multiplateforme et/ou orientée contexte- Partie II

	6-Connaissance de conception	7-Apprentissage	8-Adaptation	9-Plateformes supportées	10-Langages supportés	11-Outils
UIML	Non	Non	Statique	PC, Terminaux mobiles.	C++, CORBA, HTML, Java, J2EE, .NET, Symbian, QT, Visual Basic, VoiceXML, WML.	Liquid UI, moteurs de rendu
AUIML	Non	Non	Statique	PC, Terminaux mobiles	HTML, Java Swing	AUIML Visual Builder, plug-in Eclipse
XIML	Non	Non	Statique	PC, Terminaux mobiles, Terminaux Java	HTML, Java, WML	VAQUISTA, SEGUIA, UI generator
XForms	Non	Non	Statique	PC, Terminaux mobiles (navigateurs Web compatibles XML)	XHTML, WML	-
Plastic ML	Non	Non	Statique	PC, Terminaux mobiles	XHTML, WML, VoiceXML	ToolKit Plastic ML
AMUSING	Non	Non	Statique, Dynamique	PC, Terminaux mobiles	Java	AMUSINGs, SUNML, ISL
MOBI-D	guides de style, patrons de conception	Non	Statique	PC, Terminaux mobiles	Java	U-TEL, model editors, TIMM, interface builder
Teresa	Non	Non	Statique	PC, Terminaux mobiles	Java	TERSA, CTTE
UsiXML	Non	Non	Statique	PC, Terminaux mobiles	XHTML, Java	GrafiXML, VisiXML, SketchiXML, IdealXML, PlastiXML, ComposiXML, KnowiXML, RenderXML, ReversiXML, FlashiXML, QtkiXML, TransformiXML, Vaquita.
Comet	Non	Non	Statique, Dynamique	PC, Terminaux mobiles	Java	CamNote++
Approche de Thevenin	Non	Non	Statique	PC, Terminaux mobiles,	Java, HTML, WML, VoiceXML	ARTStudio

IV. Conclusion

Toutes les méthodes et langages présentés sont capables de générer l'interface avec différentes capacités d'intégration de la notion de multiplateforme et de multi-contexte. Nous avons distingué que la plupart des méthodes de conception orientée contexte ou également des méthodes supportant les notions de multiplateforme se focalisent sur l'adaptation pré-calculée à la conception plus qu'à l'exécution. Cette dernière peut être supportée par le retour à la conception. Dans ce cas l'interface est reconstruite de façon adéquate au nouveau contexte en revenant vers le modèle d'interface abstraite voire le modèle de tâche (en cas d'approche à base de modèles).

Nous avons vu des langages à balises visant à spécifier l'interface à un niveau plutôt abstrait ; ensuite un moteur de rendu traduit les abstraites en version finale de l'IHM. Donc pour chaque passage à une nouvelle plate-forme, il faut revenir vers le serveur, accueillant le code abstrait de l'IHM, qui génère une nouvelle version adéquate à la plateforme actuelle. Cependant toutes les entrées et les informations de l'utilisateur seront perdues. Aucune méthode orientée contexte ne prend en compte intégralement le contexte d'usage. Par exemple, UsiXML couvre une gamme encore limitée du contexte d'usage. Par exemple, dans le contexte d'environnement, il considère seulement le niveau de bruit ; par exemple, la luminosité est ignorée.

De plus, les approches sont généralement basées sur l'adaptation pré-calculée et prédéfinie par le concepteur sans mécanisme clairement défini d'amélioration de la qualité de réponse aux changements contextuels, voire d'adapter les réactions de l'adaptation aux préférences de l'utilisateur (ex. couleur du fond en cas d'une luminosité élevée).

C'est pourquoi notre objectif est de proposer une méthode de spécification et génération d'IHM plastique possédant une capacité d'adaptation dynamique applicable à l'exécution, et prenant en considération la plus grande gamme possible de chaque élément du contexte d'usage. De plus nous proposons qu'un processus d'apprentissage travaille en arrière plan du système ; son but est de préserver l'utilisabilité de l'IHM, en prenant en compte les nouveaux cas du contexte qui ne sont pas considérés pendant la génération de la base de connaissance de système, et en considérant les réactions de l'utilisateur à l'adaptation pré-calculée. La méthode proposée fait l'objet du chapitre suivant.

Chapitre III

**Proposition d'une méthode de conception
d'IHM plastique, basée sur des patrons de
conception**

I. Introduction

La recherche présentée dans cette thèse se focalise sur la conception et génération d'interfaces homme-machine (IHM) qui sont sensibles au contexte et ont la capacité de s'adapter dynamiquement au contexte d'usage (P : plate-forme, U : utilisateur, et E : environnement) tout en préservant les propriétés ergonomiques de l'IHM [Calvary *et al.*, 2001b]. Notre méthode suivra les étapes du cadre Cameleon¹⁰ (cf. chapitre II, §III.8), qui définit les étapes essentielles de développement des interfaces dans un environnement pervasif. Rappelons que Cameleon se base sur quatre principales étapes (cf. Figure III.1) : tâches et concepts, interface abstraite, interface concrète, et interface finale.

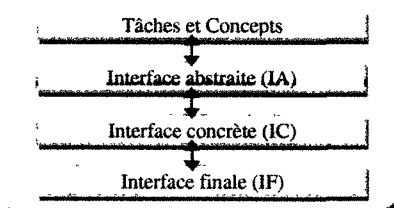


Figure III.1 : Quatre principales étapes de développement du projet Cameleon

Nous avons vu, dans le précédent chapitre, que l'adaptation au contexte d'usage, dans la plupart des méthodes, est prédéfinie par le concepteur pendant la conception. De plus, il n'y a pas éventuellement la possibilité d'évaluer la qualité de l'adaptation à l'exécution. Notre contribution vise à recalculer l'adaptation d'IHM à l'exécution afin d'améliorer l'adaptation. La Figure III.2 montre l'espace problème de la plasticité et la zone couverte par notre approche.

Nos recherches sont une extension de la version initiale d'une approche de spécification de génération des interfaces multiplateforme, proposée dans nos travaux précédents [Hariri *et al.*, 2005a, 2005b et 2006]. Dans notre méthode, nous employons la technique des patrons de conception en deux phases ; premièrement, ils sont appliqués pendant le passage à l'interface finale, puis ils sont employés lors de l'adaptation dynamique (durant l'exécution). D'une part, la technique des patrons de conception fournit en quelque sorte une connaissance réutilisable en termes de conception, permettant de trouver des solutions aux problèmes de conception récurrents, dans certains contextes d'usage. D'autre part, les patrons de conception sont considérés comme une méthode très utile pour partager des solutions éprouvées relativement à des problèmes récurrents d'IHM [Gamma *et al.*, 1994]. En nous référant à la Figure III.2, notre contribution vise à générer des IHM multimodales et à couvrir, dans la mesure du possible, la plus grande partie possible du contexte d'usage.

¹⁰ Site officiel du projet Cameleon : <http://giove.cnuce.cnr.it/cameleon.html>

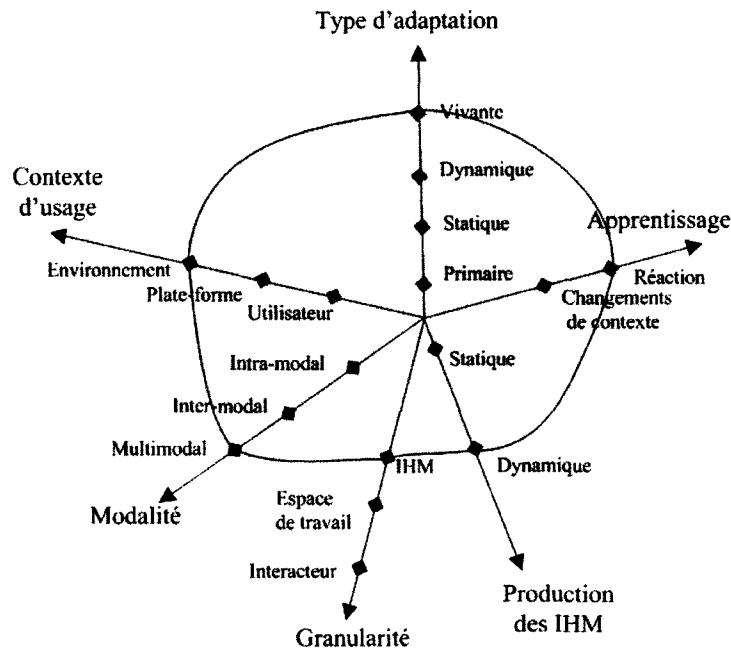


Figure III.2 : Couverture de la plasticité exprimée sur l'espace problème adapté de la plasticité (cf. chapitre 1, §IV.3.1)

La méthode est présentée au travers d'IDEF0 (Integrated DEFinition method) [Ross, 1977 ; Ang, 1999]. IDEF0 est adapté à la représentation d'activités avec leurs différentes entrées, sorties, mécanismes et interfaces de contrôle. L'environnement graphique intégré d'IDEF0 facilite la modélisation des activités constitutives de la méthode proposée dans ce mémoire, la vérification de la concordance et la traçabilité des données entre les diagrammes. IDEF0 permet une présentation de la méthode par une série de diagrammes structurés en arbres (du niveau le plus général aux niveaux les plus détaillés). Chaque activité est représentée graphiquement sur un diagramme par une boîte (Figure III.3). Celle-ci décrit une action, un processus ou une opération. Les boîtes d'un diagramme communiquent entre elles par échange de données (objets ou informations) ou par envoi de contrôles. Ces communications sont représentées sur les diagrammes par des flèches. Les données en entrée sont transformées en données de sortie. Le contrôle régit la manière dont l'activité est réalisée. Le mécanisme¹¹ représente les moyens (matériaux, logiciels, acteurs, etc.) permettant de réaliser l'activité.

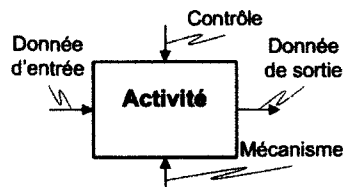


Figure III.3 : Diagramme d'activité en IDEF0

¹¹ La notion de mécanisme ne sera pas exploitée dans la suite du mémoire.

Nous présenterons tout d'abord, dans ce chapitre, les différents niveaux de l'adaptation avec un point de vue propre à nos contributions. Nous proposerons une approche originale visant la génération d'IHM plastiques à partir d'un modèle abstrait et/ou d'un modèle de tâche ; cette approche s'appuie sur les patrons de conception [Savidis et Stephanidis, 2006], en intégrant une composition basée sur les composants métier, et en incorporant des algorithmes d'apprentissage spécifiques pour être appliqués dans le domaine de l'interaction homme-machine. Dans notre méthode, nous nous interrogerons sur la manière de calculer des nouvelles réactions à des changements liés au contexte, non considérés pendant la conception. Des propositions seront faites sous l'angle de gabarit spécifique pour définir les patrons de conception utilisés dans notre méthode. Nous décrirons les caractéristiques, les sources de données, et les algorithmes d'apprentissage pour construire la base de connaissance. Un exemple de système interactif de type serveur de réservation de places de cinéma (cf. §III.2) nous accompagnera, afin d'illustrer chaque étape de la méthode. Notons que la méthode sera présentée avec la technique IDEF0 (cf. ses principes ci-dessous).

II. Niveaux de l'adaptation

La plasticité peut être envisagée à quatre niveaux, pendant la conception et l'exécution de l'interface, décrits successivement ci-dessous :

- Niveau relatif au mécanisme de l'adaptation : [Oreizy *et al.*, 1998] ont introduit deux types de systèmes auto-adaptables : les systèmes à adaptation interne (close-adaptiveness) et les systèmes à adaptation externe (open-adaptiveness). Selon cette notion, l'adaptation s'appuie soit sur un mécanisme interne, soit sur un mécanisme extérieur. Dans le premier cas, l'IHM doit être supportée par des mécanismes capables de calculer la réaction à un changement contextuel, sans nécessité de retour à la conception. L'adaptation s'effectue alors dans la plate-forme cible partiellement ou totalement, d'une part selon la capacité des ressources à disposition, d'autre part selon le changement contextuel. L'adaptation du système interactif peut être plus ou moins profonde. Quatre types de stratégies d'adaptations peuvent être envisagés [Calvary *et al.*, 2005] :
 - l'adaptation par polymorphisme : les formes des éléments de l'IHM sont changées sans aucun changement au niveau de la structure de l'IHM. Cette adaptation peut se dérouler à tout niveau de réification ;
 - L'adaptation par substitution : dans cette stratégie, au contraire de la précédente adaptation par polymorphisme, des éléments de l'IHM peuvent être remplacés par d'autres ;
 - L'adaptation par ajout : des éléments peuvent être ajoutés au système interactif. Cette stratégie permet la représentation multiple d'un même concept ;

- L'adaptation par suppression : des éléments peuvent être supprimés suite à un changement du contexte d'usage (un élément sera supprimé si sa tâche ne fait aucun sens au contexte courant).
- Niveau temporel (Figure III.4) : l'adaptation peut être effectuée à la conception de l'IHM ou à l'exécution. Dans le premier cas, l'adaptation, qualifiée de *primaire*, est réalisée pendant la conception du système interactif, avant la distribution du code exécutable. Des composants de présentation sont sélectionnés en cohérence avec la plate-forme cible, en considérant les caractéristiques de l'utilisateur et de l'environnement prévu. Dans le deuxième cas, l'adaptation, qualifiée de *vivante*, s'effectue pendant l'exécution.

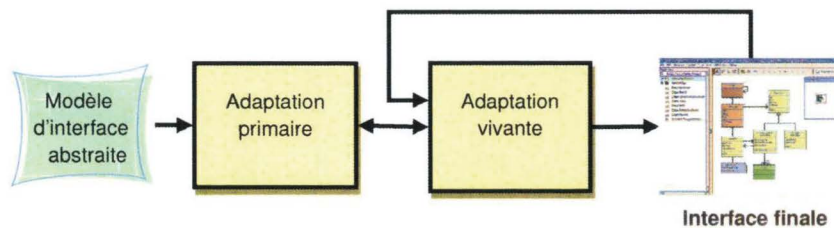


Figure III.4 : Niveau temporel de l'adaptation

- Niveau relatif à la plate-forme (cf. Figure III.5) : la plate-forme cible peut être connue ou non. Si la plate-forme est connue, alors l'interface doit pouvoir s'adapter aux caractéristiques de cette plate-forme, après les avoir vérifiées. Par exemple, dans le cas d'un PC sans souris, l'adaptation se focalise sur d'autres dispositifs d'interaction ; ceux-ci doivent pouvoir remplacer le rôle de la souris débranchée. Par contre, si la plate-forme est inconnue, le processus d'adaptation demande à la plate-forme d'envoyer ses caractéristiques et d'indiquer ses dispositifs d'interaction. Une adaptation s'effectue si la plate-forme est bien reconnue ; sinon des composants, prédéfinis par défaut et partagés entre certaines plates-formes, sont choisis pour établir une version commune de l'IHM sans spécification supplémentaire.

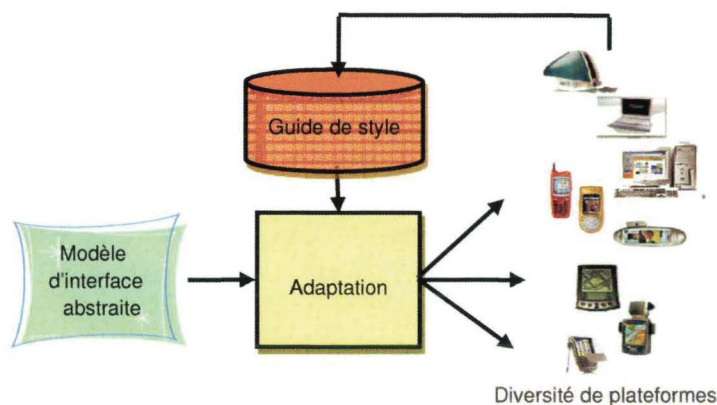


Figure III.5 : Détection de la plate-forme pendant le processus d'adaptation

- Niveau relatif à l'exécution : l'adaptation de l'IHM se déroule partiellement ou totalement, à l'exécution, selon la nature du changement sur le contexte d'usage. Par exemple, une IHM plastique est manipulée sur un PC ; soudain la souris est débranchée, l'IHM doit s'adapter partiellement à cette action. En revanche, dans certains cas, l'IHM doit pouvoir être recalculée totalement à cause d'un changement contextuel important.

Dans la partie suivante, nous allons proposer une méthode globale pour générer les IHM plastiques.

III. Méthode globale pour générer les IHM plastiques

La méthode proposée consiste à générer automatiquement une interface plastique à partir de son modèle abstrait et/ou du modèle de tâche en utilisant la technique des patrons de conception pendant le passage à l'interface finale et pendant l'adaptation dynamique (adaptation à l'exécution) [Bradbury *et al.*, 2004], tout en intégrant une architecture basée sur des composants métier. L'utilisation des composants métier vise à faciliter et augmenter la capacité de l'adaptation dynamique au contexte d'usage. L'adaptation dynamique doit permettre de préserver l'utilisabilité de l'IHM pour le contexte d'usage variable sans besoin de re-conception et ré-implémentation coûteuses. Dans ce chapitre, l'IHM est considérée comme un ensemble de composants de présentation. Les composants métier peuvent dynamiquement choisir et changer les composants de présentation de l'IHM, afin de s'adapter à un nouveau contexte, sans nécessité, donc, de retourner à l'étape de conception. Dans la méthode, l'IHM sera présentée en XML ; ensuite elle sera interprétée en langage cible de la plate-forme.

III.1 Utilisation d'IDEF0 pour la représentation des activités constitutives de la méthode

La Figure III.6 illustre les trois principaux niveaux (activités) de la méthode de génération d'interfaces plastiques :

- Un niveau abstrait (Figure III.6, A1), correspondant à celui de l'interface abstraite, l'AMXML (Abstract Model in XML) est construit à partir d'un MIA (Modèle d'Interface Abstraite) et/ou d'un modèle de tâche. Ce modèle décrit d'une part l'espace d'interaction comprenant toutes les tâches à réaliser par le système, par l'utilisateur ou par l'interaction des deux ; d'autre part il peut inclure, en fonction de la richesse du MIA d'origine, des définitions du contexte d'usage à considérer pendant la conception. Ce niveau sera détaillé dans la section III.
- Un niveau concret (Figure III.6, A2), correspondant à celui de l'interface concrète ; le noyau fonctionnel du système est construit à base de composants métier. Les composants métier intègrent des composants fonctionnels qui seront exécutés indépendamment des composants de présentation. Les composants de présentation

seront sélectionnés selon le contexte d'usage. Les détails et les différentes activités seront présentés dans la section IV.

- Un niveau final (Figure III.6, A3), correspondant à celui de l'interface finale ; c'est-à-dire l'IHM plastique effectivement manipulée. Les processus de capture du contexte et d'apprentissage coopèrent afin de faire évoluer la base de connaissance du système [Sutcliffe, 2001] dans le but de préserver l'utilisabilité de l'IHM. Nous présenterons ce niveau dans la section V.

Chaque niveau sera présenté par la suite avec ses activités détaillées ; un exemple d'un serveur de cinéma sera déroulé progressivement à chaque étape de la méthode proposée dans un but d'illustration. Nous laisserons, à l'exécution, la possibilité à l'utilisateur de modifier certaines propriétés telles que le fond, la taille de police, et de changer les composants de présentation afin de détecter ses préférences. Cette facilité laissée à l'utilisateur final, nous fournira des informations sur les réactions de celui-ci à un changement contextuel qui n'a pas été mentionné dans la base de connaissance initiale, ou au sujet d'une réaction déjà calculée suite à un changement lié au contexte d'usage, afin d'améliorer la qualité des réactions grâce aux connaissances contenues dans la base du système.

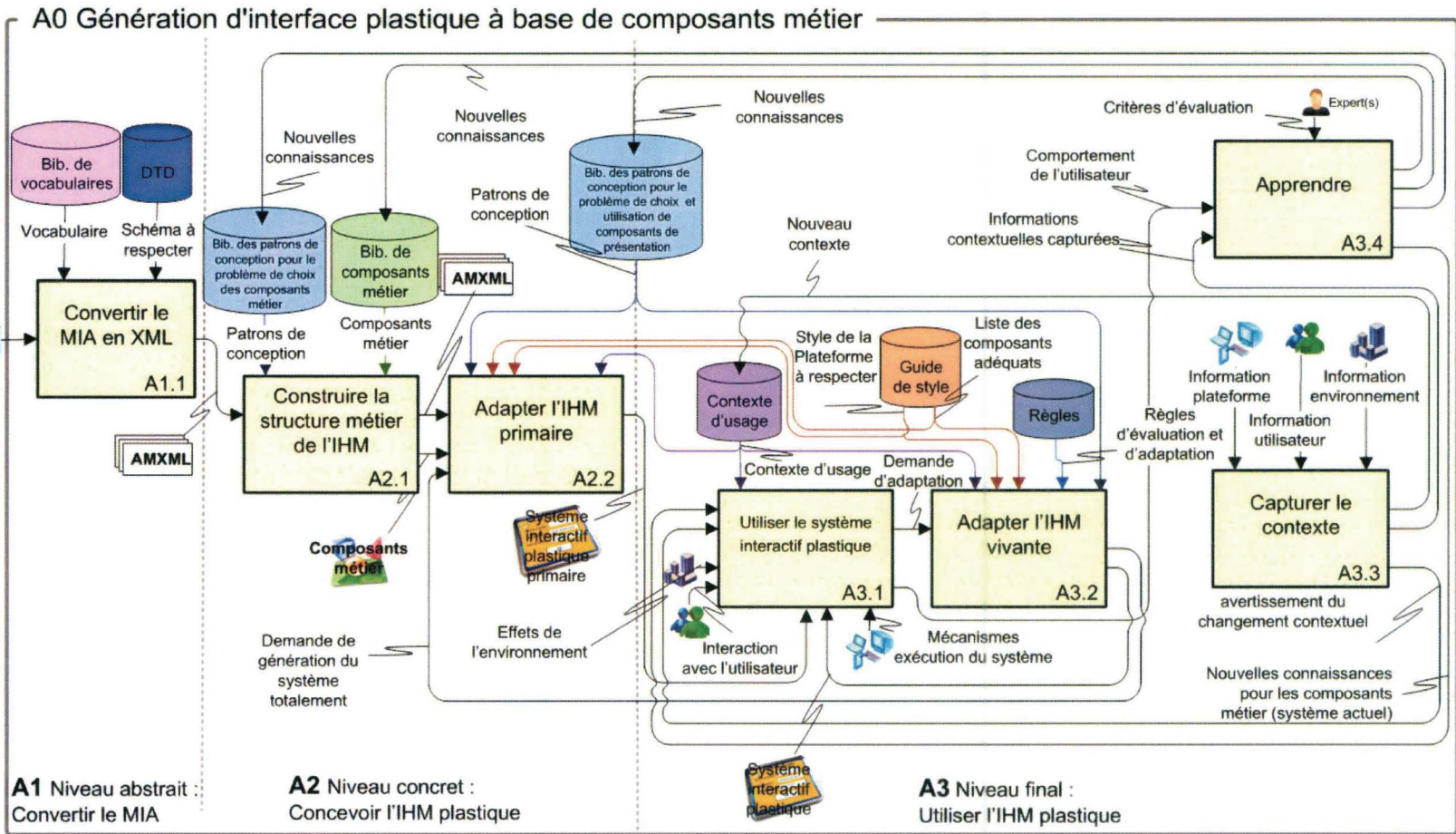


Figure III.6 : méthode globale de génération des IHM plastiques

III.2 Scénario illustratif

Afin d'illustrer la méthode, nous nous appuyons sur un système interactif de réservation de places de cinéma. Un serveur de cinéma permet aux clients de chercher un film, de trouver des informations supplémentaires sur un film et de réserver des places [Tarby, 2004]. Le contexte est celui d'un client qui veut aller au cinéma ; nous supposons que cette tâche a été analysée et décomposée en un arbre de tâches en CTT (Concur Task Trees) [Paternò, 1999]. Le modèle CTT permet de distinguer les différents types des tâches du système avec les relations chronologiques entre les tâches (cf. Figure III.7). Le système propose au client une réservation directe des places sans passer par la phase de recherche des films, ou une recherche de film selon plusieurs critères de recherche (pour s'adapter aux éventuelles préférences de l'utilisateur). Lorsque l'utilisateur trouve son film, il peut accéder à la tâche *réservation directe*.

Concernant la recherche d'un film, si le client connaît un des acteurs du film, il peut choisir l'option *recherche par acteur*. Pour effectuer cette recherche, il saisit le nom de l'acteur. Le système lance la recherche puis affiche une liste des films trouvés avec les horaires des séances. Lorsque le client en choisit un dans la liste, il accède à la tâche suivante *réservation*.

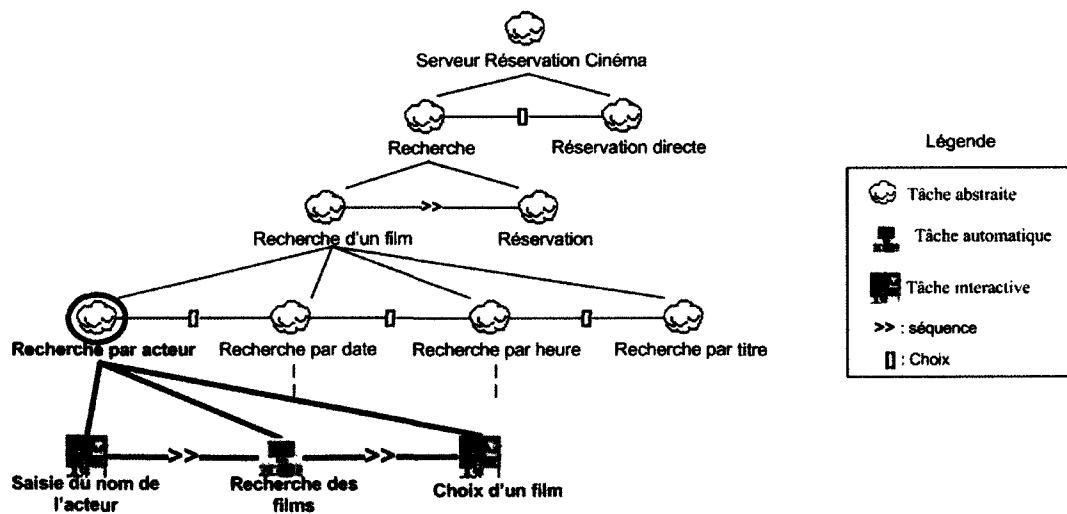


Figure III.7 : Représentation du serveur de cinéma en CTT

L'étude de cet exemple se fera en suivant les étapes de la méthode proposée. Nous nous intéresserons plus particulièrement à la tâche *recherche par acteur* (en gras dans la Figure III.7). Comme défini dans la méthode, notre hypothèse de départ est que le MIA est déjà décrit à un haut niveau d'abstraction dans une méthode de spécification de systèmes interactifs, et/ou que les tâches du système sont déjà spécifiées dans un modèle de tâche, en CTT par exemple. Dans cet exemple, à partir du modèle de tâche ci-dessus, nous pourrions constituer les descriptions abstraites de l'interface indépendamment de la modalité. Grâce au langage à balises utilisé, il sera possible de construire, de modifier et d'accéder à ce modèle pendant la conception et à l'exécution du système. Nous allons maintenant commencer à détailler le niveau abstrait de la méthode.

IV. Niveau abstrait

Il existe actuellement des langages de description d'IHM au niveau abstrait appelés UIDL (User Interface Description Language) (voir chapitre II, §III). L'objectif de ces langages est de spécifier l'IHM indépendamment de la plate-forme en utilisant le langage XML. La description de l'IHM en XML peut être instanciée pour une plate-forme, ou re- instanciée pour la renvoyer à une autre plate-forme. UsiXML (USer Interface eXtensible Markup Language) [Limboung *et al.*, 2005] permet de décrire, en plus de l'interface et des tâches, le domaine et le contexte d'usage. Ce niveau de la méthode (Figure III.6, A1) consiste en une réécriture du MIA en UsiXML (le nouveau modèle est dit AMXML). L'AMXML est un modèle abstrait de l'interface en XML qui respecte la DTD¹² (Document Type Definition) d'UsiXML avec deux ajouts : nous avons ajouté deux parties permettant d'intégrer le modèle de tâches dans l'AMXML, la première partie est spécifique aux tâches du système, et l'autre permet de définir les relations entre les tâches. L'objectif, dans cette étape, est de transcrire un modèle abstrait en AMXML, en respectant une DTD. Ce niveau inclut une bibliothèque de vocabulaires prédéfinis pour différentes représentations et une DTD à utiliser pour construire l'AMXML ; elles sont successivement décrites.

IV.1 Bibliothèque de vocabulaires

Les vocabulaires (un pour chaque type de modèle abstrait en entrée) permettent de faire le lien entre les éléments constituant le MIA et ceux définis par notre méthode. Le Tableau III.1 illustre un certain nombre de méthodes de conception de systèmes interactifs. Elles sont décrites selon les critères suivants :

- Le modèle statique de tâche permet la décomposition du système interactif en éléments significatifs appelés tâches.
- Le modèle dynamique de tâche décrit la manière dont la tâche gère et traite les données et les ressources qu'elle manipule, en fonction de son état interne (en utilisant par exemple les réseaux de petri). Il a pour but de définir les conditions d'enclenchement et de terminaison d'une tâche. Il spécifie également la répartition des données entre tâche mère et tâches filles.
- Le contexte d'usage concerne l'utilisateur avec ses préférences et habitudes, l'environnement physique et social, et la plate-forme d'interaction (cf. chapitre I, §IV.3.2).
- Les objets abstraits d'IHM ; le MIA de certains méthodes est constitué d'objets d'interaction et d'objet d'utilisateur. Un objet d'interaction décrit en termes

¹² La DTD (Document Type Definition) définit une hiérarchisation d'éléments avec des typages de ces éléments via des notions d'attribut. Le document XML se conforme à la structure syntaxique de la DTD [OMG, 2000].

d'objets, d'états, d'actions et d'enclenchement la partie de l'IHM nécessaire à l'exécution de la tâche. Un objet d'utilisateur décrit les plans d'actions de l'utilisateur (rôles de l'utilisateur) dans l'exécution de la tâche. D'autres méthodes utilisent la notion d'interacteur dans la spécification de leur MIA. En effet, la notion d'interacteur a été introduite par [Faconti et Paternò, 1990] comme une entité d'abstraction capable de réagir dans un système interactif graphique. Duke et Harrison ont généralisé la notion d'interacteur sur les systèmes interactifs en général [Duke et Harrison, 1993]. Ensuite Vanderdonckt [Vanderdonckt, 1997] a fait émerger deux niveaux d'abstraction dans la notion d'interacteur. [Thevenin et Coutaz, 1999] a recentré les interacteurs sur les concepts et tâches du domaine. Un interacteur selon lui est décrit par la liste des données qu'il est capable de représenter, la liste des tâches qu'il propose, ainsi que sa présentation graphique.

Tableau III.1 : Synthèse entre AMXML et des MIA d'autres méthodes

	Notre méthode	UsiXML (cf. chapitre II, §III.8.2)	Méthodes MBD		TERESA (cf. chapitre II, §III.8.1)
			TOOD [Tabary, 2001]	DIANE+ [Tarby et Barthet, 1996]	
Tâches statiques	Structure basée sur des composants métier	Basé sur un modèle CTT	Modèle statique de tâche	Modèle statique de tâche	Basé sur un modèle CTT
Tâches dynamiques	Composants fonctionnels	Basé sur un modèle CTT	Modèle dynamique de tâche	Modèle dynamique de tâche	Basé sur un modèle CTT
Contexte de plate-forme	Oui	Oui	Utiliser UIML	-	Oui
Contexte d'utilisateur	Oui	Oui	Ressources humaines	Modèle de dialogue	Oui
Contexte environnemental	Oui	Oui	-	-	-
Objets de l'IHM abstraite	Conteneurs abstraits	Conteneurs et composants abstraits	Objets d'interaction et d'utilisateur	Objets d'interaction et d'utilisateur	Interacteurs

La comparaison, en termes de spécification de l'abstraction des IHM, illustre que toutes les méthodes, mentionnées dans le tableau, sont capables de générer l'interface avec différentes capacités d'intégrer des abstractions relatives à la plasticité de l'IHM.

Notre méthode propose l'intégration du modèle de tâches dans l'AMXML grâce aux balises supplémentaires ajoutées au MIA d'UsiXML. Cependant, les méthodes UsiXML [Limboung *et al.*, 2005] et TERESA [Mori *et al.*, 2002] sont basées sur le modèle de tâches CTT [Paternò et Santoro, 2002 ; Mori *et al.*, 2004], de sorte que le modèle CTT puisse être attaché au MIA aux étapes de génération d'IHM ; alors le modèle CTT doit être maintenu afin de le réutiliser en cas d'une demande d'adaptation. Notre méthode a l'avantage de l'intégration du modèle de tâches dans un seul modèle MIA, ce qui facilite la

spécification des abstractions de l'IHM plastique relatives aux tâches du système. Par exemple, chaque tâche du système de serveur de cinéma est associée à un ou plusieurs *conteneurs abstraits*. Ceux-ci seront spécifiés selon la nature des tâches.

Pour ce qui concerne le contexte de plate-forme, TOOD [Tabary, 2001 ; Mahfoudhi *et al.*, 2005] adopte UIML dans l'étape de génération de l'IHM afin d'ajouter la notion de multiplateformes aux IHM générées [Hariri *et al.*, 2005b].

De plus, nous pourrions considérer les ressources humaines, dans TOOD, comme une partie du contexte d'usage (en particulier le contexte de l'utilisateur (cf. chapitre I, §IV.3.2)). Diane+ [Tarby et Barthet, 1996] est une méthode à base de modèle dédiée particulièrement aux systèmes d'information. Diane+ et TOOD ont réussi récemment à supporter au moins partiellement la notion de multiplateforme.

Nous avons distingué dans le chapitre II que la plupart des méthodes de conception orientée plasticité ou supportant la notion de multiplateformes se focalisent sur l'adaptation à la conception plus qu'à l'exécution. Cette dernière peut être supportée par le retour à la conception. Dans ce cas l'interface est reconstruite selon les caractéristiques du nouveau contexte. Au contraire, notre méthode essaye d'effectuer l'adaptation à l'exécution ; cependant si cette dernière ne peut pas appliquer les modifications nécessaires pour l'IHM, et dans ce cas seulement, il y a retour à la conception pour effectuer l'adaptation et la reconstruction de l'IHM.

L'AMXML est construit à l'aide d'un ensemble de *conteneurs abstraits* qui seront concrétisés par l'ajout de facettes de présentation des composants métier correspondant au système généré. Les MIA de la méthode TOOD et de Diane+ possèdent une possibilité de modélisation du (des) rôle(s) de l'utilisateur avec ses activités, ainsi que des objets d'interaction définissant les tâches d'interaction avec l'utilisateur. La grande différence entre AMXML et son parent MIA d'UsiXML est que les tâches du système sont spécifiées avec leurs relations dans un seul modèle. Ensuite les tâches seront traduites en composants métier pendant la construction du noyau fonctionnel. UsiXML n'intègre pas les descriptions sur des tâches dans son MIA, alors qu'un modèle de tâches CTT doit être attaché au MIA d'UsiXML. D'ailleurs, UsiXML couvre une gamme limitée du contexte d'usage ; par exemple dans le contexte d'environnement, il considère seulement le niveau de bruit, mais la luminosité, par exemple, est ignorée pour le moment. Au contraire, nous essayerons de prendre en considération une large gamme de chaque facette du contexte d'usage.

IV.2 DTD d'AMXML

AMXML respecte une DTD qui fournit un schéma complet de la structure XML utilisée dans notre méthode. La DTD d'AMXML (cf. Figure III.8) commence avec la balise `<amxml>`. Cette balise est composée de quatre parties principales :

- La balise `<taskModel>` décrit hiérarchiquement les tâches du système avec leurs types (manuelle, interactive, automatique, abstraite), et détermine les *conteneurs*

abstracts (interface abstraite) attachés aux tâches. Cette balise est construite selon les tâches décrites dans le modèle de tâche d'origine ; les relations entre les tâches (activation, désactivation, choix, simultanéité, etc.) sont réalisées grâce à la partie *<relationships>* ;

- La balise *<guiModel>* est composée de *conteneurs abstraits <abstractcontainer>* qui sont attachés aux tâches principales (par exemple, un *conteneur abstrait* est défini pour la tâche *recherche par acteur*). Les conteneurs peuvent inclure des composants abstraits associés aux sous-tâches (par exemple, un *composant abstrait* est défini dans l'AMXML pour la sous-tâche *saisie du nom de l'acteur*) ;
- La balise *<contextModel>* comprend les contextes d'usage prévus selon le triplet *<plate-forme, utilisateur et environnement>*, définis dans cette partie ;
- La balise *<resourceModel>* indique toutes les ressources de l'IHM ; ces ressources peuvent être exprimées textuellement ou correspondre à des données (image, vidéo, son, etc.). Un conteneur peut posséder différentes ressources pour des contextes d'usage.

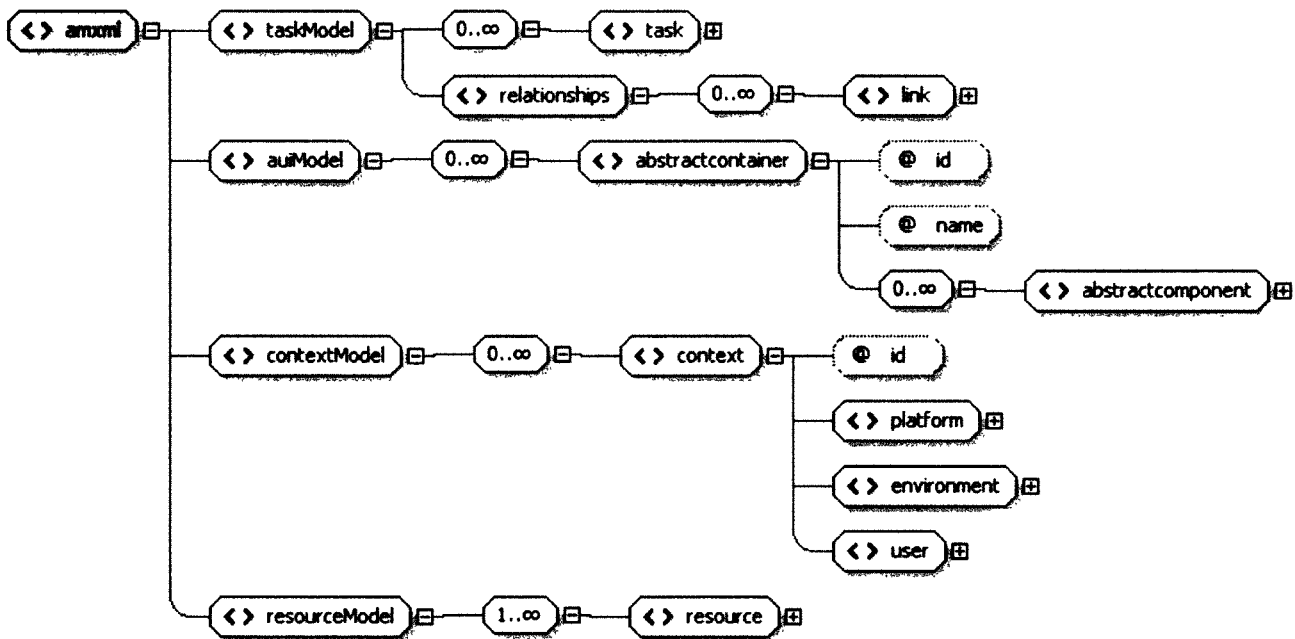


Figure III.8 : Schéma de l'AMXML (la DTD complète se trouve en annexe A)

IV.3 Conversion du MIA en AMXML

La Figure III.9 détaille l'activité A1.1 "Convertir le MIA en AMXML" visible en Figure III.6, dans la méthode globale. La conversion débute par l'identification de la représentation du MIA. Cette identification prend en entrée un MIA d'une méthode de spécification et de conception des systèmes interactifs (ex. TOOD, DIANE+, etc.).

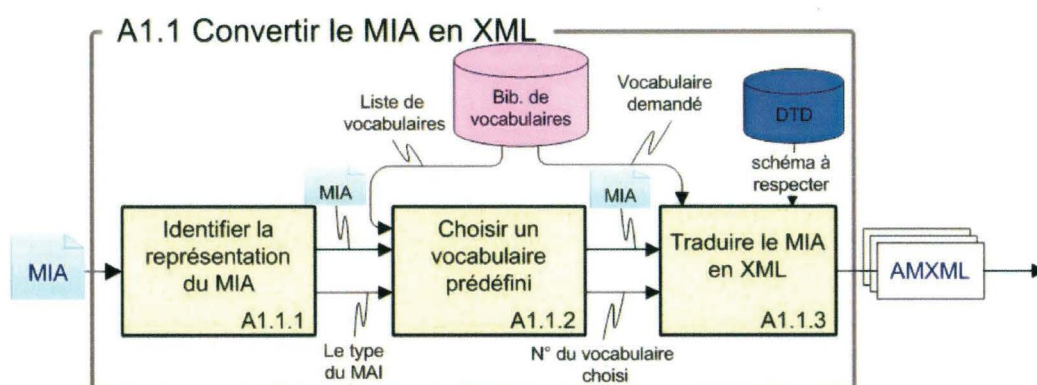


Figure III.9 : Convertir le MIA en AMXML

L'identification permet de choisir un vocabulaire à utiliser pour traduire le MIA en AMXML. La traduction s'appuie sur une DTD et sur une bibliothèque de vocabulaires assurant les correspondances du MIA d'origine vers l'AMXML. Ce dernier est construit à partir des informations récupérées du MIA d'origine ; ensuite, il sera complété automatiquement par des informations génériques. Enfin, le concepteur du système peut modifier ou spécifier des parties de l'AMXML. Grâce au langage à balises utilisé pour décrire l'abstraction de l'IHM plastique, il est possible d'accéder à l'AMXML et de le modifier pendant la conception et à l'exécution (l'AMXML est distribué avec le code exécutable vers la plate-forme). Nous proposons six règles permettant de construire l'AMXML à partir du modèle d'interface abstraite et/ou du modèle de tâche :

1. La partie `<taskModel>`, contient toutes les tâches du système et les liens entre elles. Elle est basée sur un modèle de tâche. Les tâches sont décrites grâce à la sous-balise `<task>`. Les opérateurs temporels liant les tâches (activation, concurrence,...), sont intégrés dans la partie `<relationships>` (une sous-balise `<link>` pour chaque opérateur).
2. Les *conteneurs abstraits* `<abstractcontainer>` sont associés à la tâche racine, et aux tâches mères (qui contiennent elles-mêmes des sous-tâches mères et/ou terminales). Les conteneurs se nomment de la même manière que les tâches leur correspondant. En cas d'existence d'un modèle d'interface d'origine, les *conteneurs abstraits* sont construits à partir de ceux correspondant dans le modèle d'interface d'origine.
3. Les *composants abstraits* `<abstractcomponent>` sont associés aux tâches automatiques et interactives, en plaçant ces composants dans le *conteneur abstrait* de leur tâche mère. Les conteneurs se nomment de la même manière que les tâches leur correspondant. En cas d'existence d'un modèle d'interface d'origine, les *composants abstraits* sont construits à partir de ceux correspondant dans le modèle d'interface d'origine.
4. Les *conteneurs abstraits* et les *composants abstraits* sont organisés en respectant la hiérarchie des tâches dans le modèle de tâche.

5. La partie *<context>* est construite en spécifiant des valeurs génériques s'il n'y a pas d'informations contextuelles à importer. Ces dernières peuvent être extraites du modèle d'interface d'origine (par exemple, la langue de l'utilisateur est celle officielle du pays si celle-ci est unique, la luminosité est considérée comme moyenne, etc.), ou éventuellement des informations provenant du modèle de tâche (par exemple, telle tâche s'exécute sur un PDA avec écran tactile).
6. Les *conteneurs abstraits* ne contenant aucun composant ou conteneur sont supprimés.

Dans le système du serveur de cinéma, une description abstraite de l'interface plastique spécifique à la tâche *recherche par acteur*, sera obtenue indépendamment de la modalité, à partir d'un modèle abstrait d'interface spécifié dans une autre méthode classique, en utilisant un vocabulaire adapté. Un extrait de l'AMXML dans lequel l'interface plastique est spécifiée est illustré dans la Figure III.10.

Nous proposons que chaque tâche abstraite du modèle de tâche ait un *conteneur abstrait*. Donc la partie *auimodel* de l'AMXML généré, contient trois *conteneurs abstraits* spécifiques aux trois sous-tâches : *saisie du nom de l'acteur*, *recherche des films*, et *choix d'un film*. Par exemple, la tâche dont l'identifiant est « *t_search* » génère une liste des films trouvés, et les affiche sur son propre conteneur afin de permettre à l'utilisateur de choisir un film afin de réserver des places par la suite.

Les relations et les échanges de données entre les tâches sont définis dans la partie *relationships*. Par exemple, dans la balise « *l_input_search* » de la partie *relationships*, la tâche « *t_input* » (définie comme *source*) envoie le nom de l'acteur à la tâche « *t_search* » (définie comme *target*).

Dans l'exemple, deux contextes d'usage sont prédéfinis ; dans le premier contexte, par défaut, un utilisateur est défini ayant 16 ans, parle anglais, et en état pressé. Des informations sur l'âge pourraient être employées dans le filtrage des films trouvés (dans la mesure où certains films incluent des scènes déconseillées aux personnes d'un certain âge). Dans l'autre contexte, un utilisateur français est décrit. Selon les langues d'utilisateur décrites dans les contextes prédéfinis, les ressources des composants de présentation sont spécifiées (en particulier les ressources textuelles) ; deux ressources textuelles sont définies pour les deux utilisateurs anglais et français, avec la possibilité de définir plus de ressources pour d'autres catégories d'utilisateur.

Nous allons passer maintenant au niveau concret de la méthode en détaillant chacune des activités.

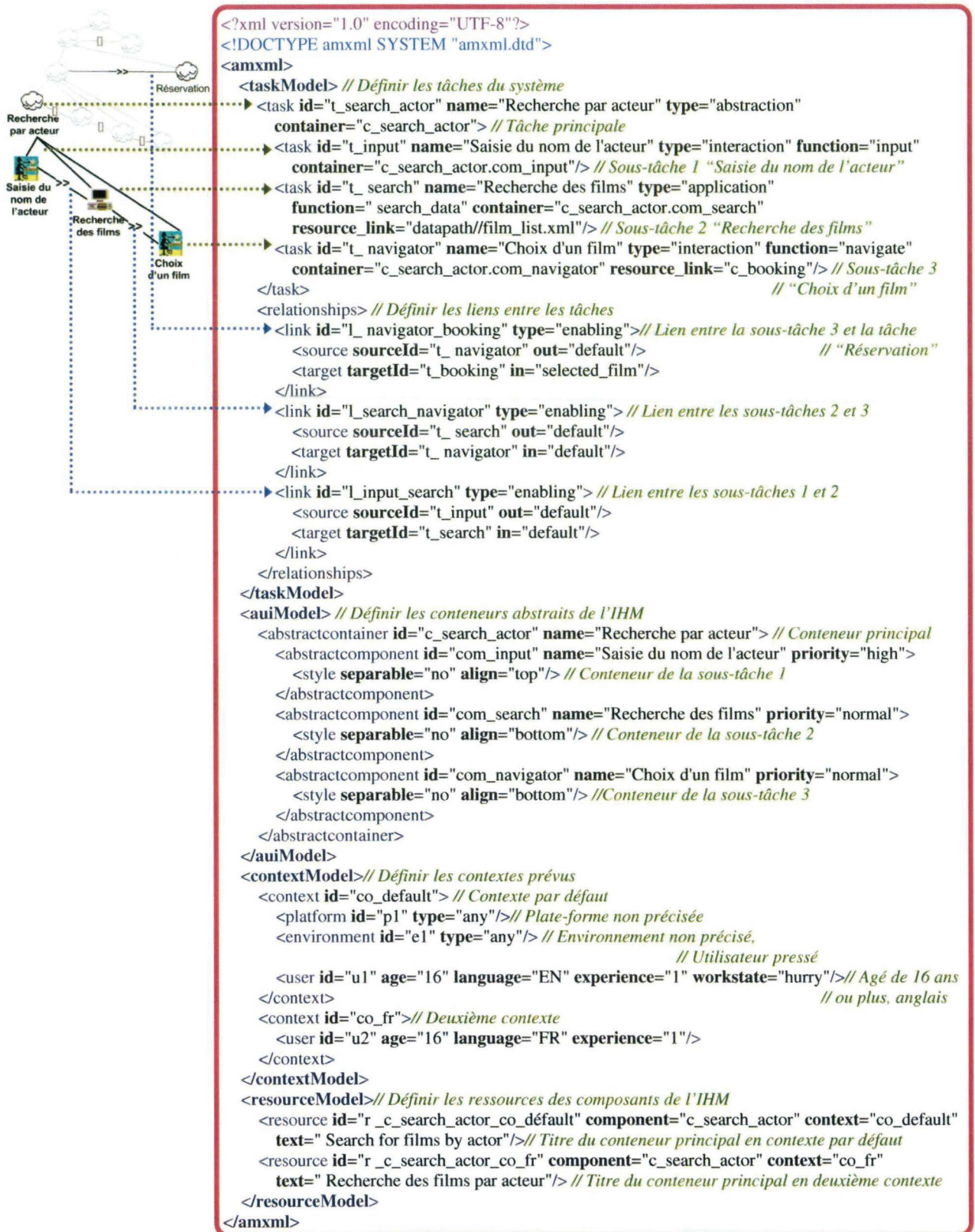


Figure III.10 : AMXML de l'IHM « recherche par acteur »

V Niveau concret

Au niveau concret (cf. Figure III.6, A2), nous proposons que la réification de l'IHM plastique (cf. Figure III.6, A2) passe d'abord par la construction du noyau fonctionnel et ensuite qu'une première adaptation de l'IHM s'effectue avant la distribution du système à la plate-forme cible. Les règles d'adaptation peuvent avoir la forme « Action → Réaction » [Thevenin et Coutaz, 1999] : une action indique un changement qui apparaît dans le cadre de l'utilisation, et une réaction indique les procédures à impliquer. D'autres approches ont adopté l'adaptation du type « Situation → Réaction » [Grolaux *et al.*, 2002] : la situation indique un état du contexte d'usage à considérer par l'IHM, une réaction est réalisée en conséquence de cet état.

[Calvary *et al.*, 2003] ont proposé l'adaptation du type « Action * Condition → Réaction » où les réactions aux actions d'utilisation peuvent être variées en fonction des conditions des éléments du contexte d'usage ; nous avons adopté ce dernier type d'adaptation dans notre méthode parce que la forme d'adaptation « Action * Condition → Réaction » est, selon nous, la plus adéquate dans notre cas où les actions d'utilisation doivent être considérées par le processus d'adaptation en prenant en compte le contexte d'usage courant.

Les règles d'adaptation sont définies initialement à la conception par le concepteur. Ensuite, les règles peuvent être améliorées, ou de nouvelles règles peuvent être ajoutées pendant l'exécution lors de l'apprentissage (abordé en §V.3). Les règles peuvent utiliser les opérateurs de (dé)composition proposée par [Lepreux et Vanderdonckt, 2006a ; Lepreux *et al.*, 2006b]. Ce niveau utilise (i) une bibliothèque de composants métier et (ii) deux bibliothèques de patrons de conception, successivement décrites.

V.1 Bibliothèque de composants métier

La notion d'intelligence artificielle [Russell *et al.*, 2002], est intégrée dans les composants métier. L'intelligence artificielle est basée sur des principes spécifiques, tels que des structures de données, des algorithmes, des langages et des techniques de programmation, afin de prendre des décisions automatiquement en référence à une expertise du domaine (acquise auprès d'un ou plusieurs experts). Les types de décisions sont variés en fonction du domaine d'application.

Le système expert [Ignizio, 1991] (ou système à base de règles) fait partie de l'intelligence artificielle. Rappelons qu'il reproduit le comportement d'un expert humain accomplissant une tâche décisionnelle dans un domaine précis. Par exemple, le système expert peut être conçu pour résoudre des problèmes de classification ou de décision. Il est composé généralement de deux parties indépendantes (cf. Figure III.11) : une base de connaissances composée de règles qui modélisent la connaissance du domaine considéré, et une base de faits qui contient les informations concernant le cas à traiter. Un moteur d'inférence est capable de raisonner à partir des informations contenues dans la base de connaissance, par

exemple pour effectuer des déductions, et produire des décisions relativement aux faits conçus.

Dans notre cas, les décisions du système expert (minimal) intégré dans le composant sont relatives à l'adaptation de l'IHM, en considérant que la base de faits contient des actions (ou événements) liées au changement du contexte, ou des actions de la part de l'utilisateur, et des conditions liées au contexte d'usage. Par exemple, une action peut être : "la luminosité a augmenté", et une condition : "si la plate-forme est un PDA".

Les composants métier utilisés dans notre méthode possèdent une base de connaissance intégrée (cf. Figure III.12). Nous avons proposé l'idée d'intégration d'une base de connaissance afin d'aider à prendre des décisions à l'exécution, en particulier les décisions de choix et de changement de la facette de présentation, face au changement du contexte d'usage. Les composants peuvent avoir leur propre moteur d'inférence, ou peuvent utiliser un moteur externe en commun entre les composants (embarqués ou non sur le système). Le choix pourra se faire selon les contraintes temporelles liées au domaine d'application.

L'intégration de la base de connaissance vise à :

- diminuer le temps d'adaptation à l'exécution (sans aucune intervention de l'utilisateur ou du concepteur au niveau des décisions à prendre),
- réduire la charge du réseau de connexion (dans la mesure où la charge peut dépendre du rechargement du système à partir du serveur, à chaque adaptation effectuée à la conception),
- et optimiser le temps de réponse vis-à-vis du changement contextuel (le temps à attendre entre le moment du changement du contexte et l'application de la réaction).

Dans notre méthode, un mécanisme s'appuyant sur cette base joue le rôle de décideur concernant le choix de composants de présentation en fonction du contexte d'usage. De plus, nous proposons que la base travaille en synchronisation avec les patrons de conception ; les patrons devront pouvoir proposer des solutions conceptuelles, concernant par exemple l'arrangement des composants de présentation sur l'IHM, ou le changement de l'endroit de la facette de présentation selon l'importance de la facette (cas d'une plateforme graphique).

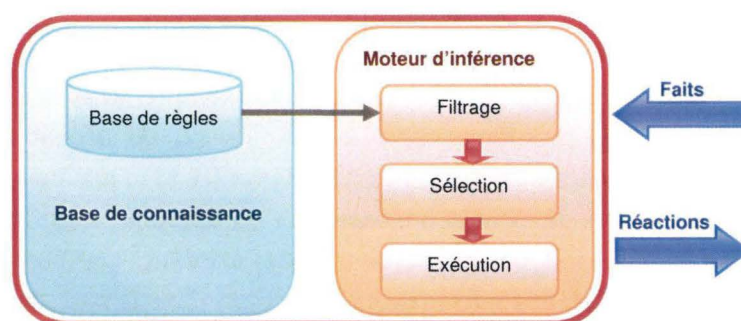


Figure III.11 : Architecture d'un système à base de règles [Durkin, 1994]

Lors d'un changement de contexte, la plate-forme peut changer alors que l'utilisateur reste le même (notion de migration d'IHM). Dans ce cas, les composants métier correspondant au système doivent assurer une adaptation dynamique (nouveau choix des composants de présentation) sans perdre les données relatives à l'utilisateur courant grâce à leurs mémoires cache¹³. L'intégration d'une mémoire permet de conserver les informations et les entrées de l'utilisateur. Ensuite les données conservées sont récupérées et placées dans les nouveaux composants de présentation.

Les règles contenues dans la base de connaissance peuvent nécessiter d'être améliorées ; pour ce faire, des techniques d'apprentissage automatique peuvent être appliquées pour la base de connaissance à l'exécution (cf. §V.3, page 113).

La bibliothèque de composants métier utilisée contient la liste des composants métier qui sont classifiés selon le domaine d'utilisation (login, gestion de compte bancaire, recherche, etc.) afin de faciliter leur utilisation pendant la génération du système.

La Figure III.12 illustre un exemple de composant métier « intelligent » (en intégrant une base de connaissance) : appelé "*choix*", il possède :

- une interface d'entrée/sortie fonctionnelle (IF), une autre pour la présentation (IP) du composant, une base de connaissance permettant de choisir sa facette de présentation en fonction du contexte ;
- une mémoire cache qui sera utilisée par les composants de présentation pour stocker les informations et les entrées de l'utilisateur ;
- deux composants fonctionnels *CF1* et *CF2*. Les deux composants peuvent travailler en collaboration entre eux, ou tous les deux peuvent proposer la même fonctionnalité, mais dans deux contextes différents, en prenant en compte par exemple la puissance de traitement de la machine.
- deux composants de présentation *CP1* (de type *DropDownListBox*¹⁴) et *CP2* (de type *ListBox*¹⁵) correspondant à deux différents contextes (*C1*, *C2*). Pendant l'adaptation, un champ spécifié pour le composant métier "*choix*" et contenant une

¹³ La mémoire cache des composants métier est un espace dont les ressources de stockage pourrait être utilisées par les composants de présentation à l'exécution. Dans cet espace réservé, des informations et des entrées de l'utilisateur peuvent être sauvegardées ; elles peuvent être transférées vers une nouvelle plate-forme, puis seront récupérées (notion de migration d'IHM [Bandelloni et Paternò, 2004]).

¹⁴ *DropDownListBox* : ce type est similaire au type *ListBox* mais les items de la liste sont repris sous la forme d'un menu déroulant. Il permet de gagner de l'espace surtout dans un PDA ou téléphone portable.

¹⁵ *ListBox* : affiche un choix d'item dans une liste d'items afin d'en sélectionner un. Tous les éléments de la liste sont organisés dans un cadre pouvant posséder une barre de défilement si la liste d'éléments est plus longue que la taille du cadre.

liste d'éléments, peut être projeté vers une *DropDownListBox* et vice versa selon le contexte.

- Et une base de connaissance (BC) qui sera utilisée par un moteur d'inférence externe. Ce dernier donne la décision de choix de la présentation (dans cet exemple de composant métier, le choix s'effectue en fonction de l'espace d'affichage disponible pour la facette de présentation).

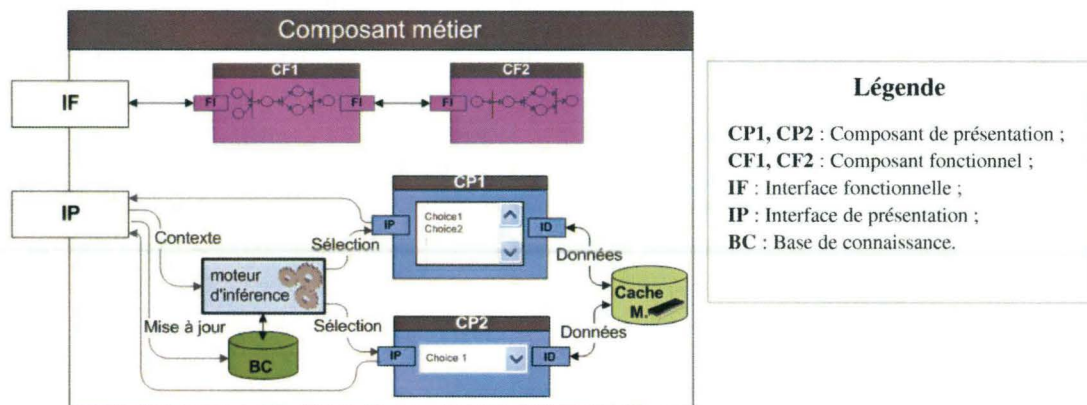


Figure III.12 : Exemple de composant métier (cas où le moteur d'inférence est interne au composant)

V.2 Patrons de conception

La notion de patron de conception est intégrée dans notre démarche à deux niveaux : au niveau de la construction de la structure métier¹⁶ du système, et à celui de la conception et de l'adaptation de l'IHM. Nous avons adopté la notion de patron dans notre méthode, afin de fournir des solutions aux problèmes récurrents de conception relatifs à l'adaptation de l'IHM.

V.2.1 Proposition d'un gabarit en vue de l'adaptation

Le gabarit, proposé pour accompagner la méthode, est présenté sous la forme du Tableau III.2. La partie originale dans cette structure est le *Contexte d'utilisation* qui est la condition d'exécution du patron. De plus, le champ de *Déclenchement* est indispensable pour le processus d'adaptation afin de savoir si la solution est applicable à l'exécution et/ou à la conception. Les patrons sont transcrits en XML afin de faciliter leur stockage et leur transmission [Gamma *et al.*, 1994]. Ensuite la solution peut être sous forme de code à interpréter par un moteur d'interprétation, pendant l'appel du patron. Les paramètres des solutions apportées par les patrons, les conditions d'utilisation des patrons, peuvent être à tout moment modifiés par l'expert.

¹⁶ Structure métier : Structure composée de composants métier qui sont reliés entre eux grâce à une glue (cf. §V.5).

Nous avons implémenté un outil de développement de patrons de conception nommé PaDev (PAttern DEVeloper) (il est décrit en annexe C). Cet outil vise à poursuivre le développement structuré et intégré des patrons de conception en suivant le gabarit proposé ci-dessus.

Tableau III.2 : Gabarit permettant de définir les patrons associés à la méthode (la DTD complète se trouve en annexe B)

Id	Ce champ permet d'identifier chaque patron. Il est utilisé pour stocker le patron dans une base et permet de le retrouver aisément (cf. Figure III.13, Partie « <i>id</i> »).
Nom	Le nom du patron est significatif et correspond au problème à résoudre (cf. Figure III.13, Partie « <i>name</i> »).
description	La description exprime de manière textuelle le but du patron. Elle permet aux développeurs d'avoir une idée sur le patron avant de l'utiliser ou de l'améliorer (cf. Figure III.13, Partie « <i>description</i> »).
Contexte d'utilisation	Le contexte d'utilisation du patron concerne, dans notre cas, l'étape d'adaptation dans laquelle se situe le système. Les patrons sont classifiés selon leur contexte d'utilisation en trois catégories : patrons à appeler à la conception, ceux à utiliser à l'exécution, et ceux qui peuvent être appliqués à la conception et à l'exécution (cf. Figure III.13, Partie « <i>utilizationcontexts</i> »).
Contexte d'usage	Le contexte d'usage donne des informations sur la plate-forme, l'environnement et l'utilisateur, permettant d'utiliser le patron (cf. Figure III.13, Partie « <i>usecontext</i> »).
Déclenchement	Le déclenchement donne une idée du niveau de granularité du patron. Si celui-ci intègre du code implémentable, il pourra être exécuté ou déclenché automatiquement. Au contraire, si le patron intègre des solutions d'un niveau de granularité plus élevé, le patron ne pourra se déclencher sans l'intervention humaine (cf. Figure III.13, Partie « <i>launching</i> »).
Solution	Cette partie intègre la solution qui est apportée au problème dans le contexte d'utilisation et d'usage donné. La solution peut être de type modélisation ou plus finement décrite (par ex. code exécutable) (cf. Figure III.13, Partie « <i>solution</i> »).
Collaboration	Cette partie permet de faire appel à un autre patron concernant un sous-problème du problème décrit ici (cf. Figure III.13, Partie « <i>collaboration</i> »).

La Figure III.13 illustre un exemple de patron de conception suivant le gabarit proposé ci-dessus. Le patron, ayant l'identification P10201, est applicable à l'exécution, sur les plateformes graphiques. Le patron peut être déclenché automatiquement par le processus d'adaptation vivante. Il recalcule la taille et la localisation considérées comme optimales pour l'IHM, suite à un changement dans le contexte de la plate-forme (en particulier la taille de l'écran d'affichage).

Le patron essaye aussi de trouver les meilleures tailles et localisations pour les composants de présentation, dans le cas où le taux du changement de la taille de l'écran varie entre 0.5 et 1.3 (le processus d'apprentissage à l'exécution ou le concepteur, peut modifier ces chiffres). Par contre si le changement est de moins de 0.5, le patron appelle le patron P10225 qui est le responsable de l'action consistant à diviser les fenêtres en sous-fenêtres afin de bien investir l'espace d'affichage disponible. Sinon, il appelle le patron P10226 qui trouve des solutions de manière à fusionner des fenêtres ; ce patron utilise des opérateurs de composition d'IHM [Lepreux et Vanderdonckt, 2006a ; Lepreux *et al.*, 2006b]. Cette

fusion a pour but de bien étendre l'IHM sur tout l'espace d'affichage changé. La Figure III.13 illustre l'exemple du patron représenté en XML, suivant le gabarit proposé ci-dessus.

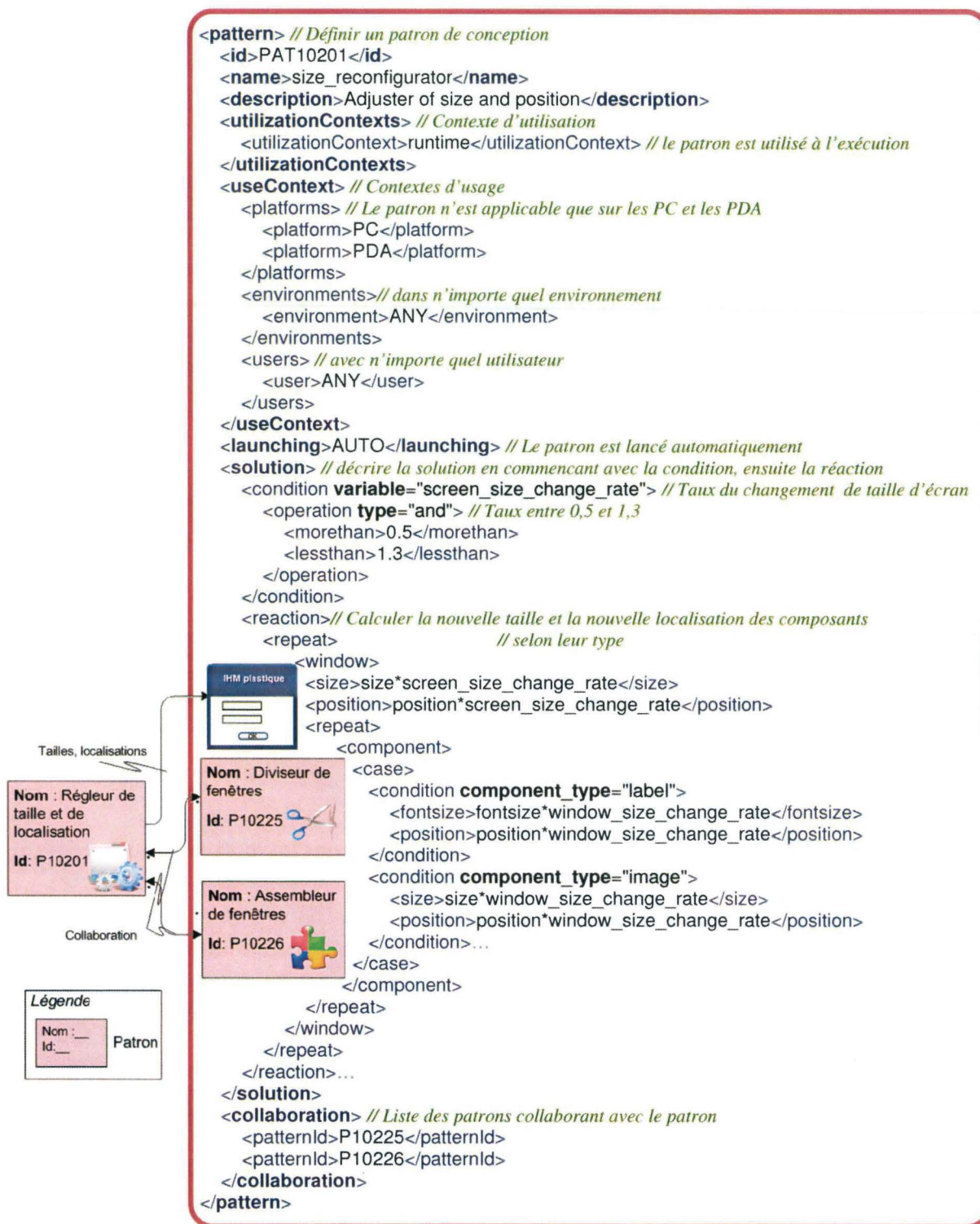


Figure III.13 : Représentation en XML du patron "régleur de taille et de localisation"

L'application des patrons de conception peut être présentée en trois étapes [Sining *et al.*, 2004]. Tout d'abord, une facette de présentation W qui constitue une partie de l'IHM est identifiée : $W \subseteq \text{IHM}$. Ensuite un patron P approprié à W est choisi. Le patron P sera ajusté, par la méthode A , selon le contexte d'usage C ayant l'objet de spécifier une instance du patron S . Donc $A(P, C) = S$. Enfin l'instance S du patron P sera appliquée sur la facette W afin de modifier ses structures de conception. Donc $F(W, S) = W'$ qui correspond à la facette W modifiée, par l'instance S .

L'application des patrons dans la méthode peut rencontrer deux problèmes majeurs identifiés ici :

- Les dépendances non résolues. A titre d'exemple, supposons que le patron A dépend du patron B , si le bon fonctionnement de A nécessite la présence de B . Une modification probable du patron B , ou le changement de la stratégie de résolution peut risquer de perdre l'utilité de la solution portée par le patron A , ou de l'impossibilité d'utiliser le patron A . Une dépendance n'est pas résolue lorsqu'un patron est modifié sans l'avertissement des patrons dépendants.
- Un autre conflit fait référence au cas où deux patrons ou plus peuvent être utilisés pour un même problème. Une solution basée sur la priorité des règles peut être actionnée pour traiter ce problème.

V.2.2 Bibliothèques de patrons de conception

Notre approche s'appuie, aux niveaux concret et final, sur deux bibliothèques de patrons de conception (visibles en Figure III.6):

- *Bibliothèque des patrons de conception pour le problème de choix des composants métier* : elle fournit au système des patrons de conception qui ont pour but d'aider à choisir les composants métier convenables à un modèle de tâche. Ceux-ci préconisent aussi des solutions de composition. La Figure III.14 illustre deux compositions possibles pour les trois fonctionnalités (*Copier*, *Couper*, et *Coller*). A gauche, l'architecture métier est composée d'un seul composant métier composite "*edition*" qui inclut les trois composants métier nécessaires pour effectuer les trois fonctionnalités ; l'avantage de cette composition est que la liaison des composants métier est déjà effectuée ; donc ce composant composite métier ne nécessite pas beaucoup de travail pour l'intégrer dans la structure métier du système. Cependant la composition interne des trois composants ne peut pas être modifiée, et nous sommes obligés de garder toujours les trois composants : par exemple, si nous n'avons pas ou plus besoin du composant métier *Couper*, nous ne pouvons pas l'éliminer. La composition à droite, a trois composants métier correspondant aux trois fonctionnalités du système, qui sont sélectionnés et reliés entre eux. L'avantage de cette composition vient des possibilités de modification de la composition, de suppression des composants qui ne sont plus utilisés, et d'ajout de nouveaux composants si besoin par composition.

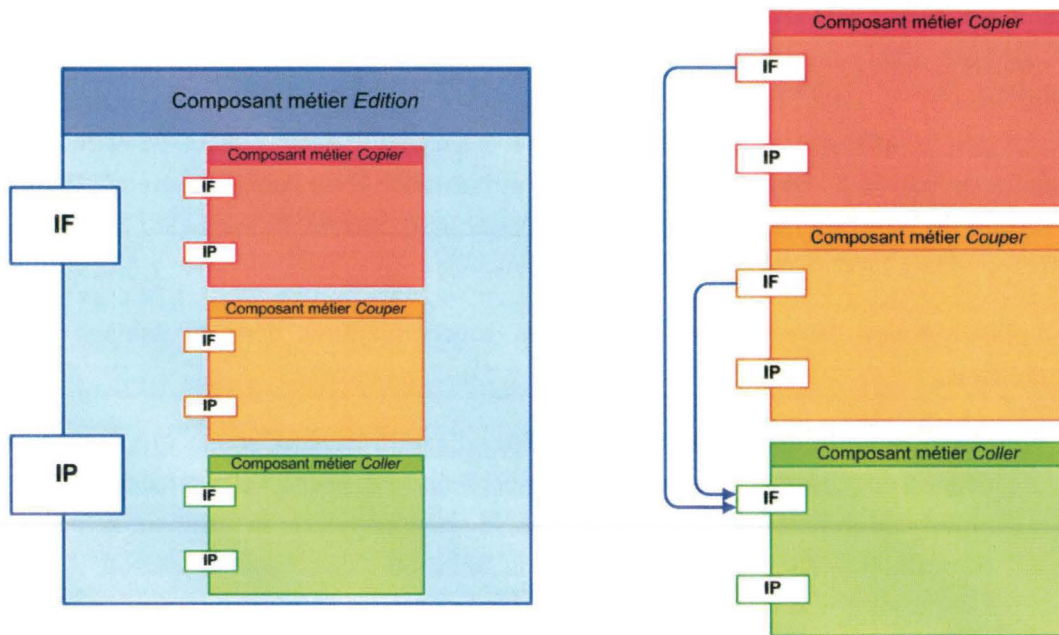


Figure III.14 : Deux compositions métier possibles

- *La Bibliothèque des patrons de conception pour le problème du choix et de l'utilisation de composants de présentation* : elle fournit des patrons de conception qui préconisent des solutions pour choisir et assembler les composants de présentation lors de l'adaptation primaire (niveau concret) et de l'adaptation vivante (niveau final). Notons que certains patrons de conception sont spécifiques à l'adaptation vivante et ne sont pas applicables à l'adaptation primaire. Par exemple, le patron qui donne la solution optimale au changement du contexte de l'utilisateur n'est utilisable qu'à l'exécution. L'intégration des patrons, de deux niveaux, dans une seule bibliothèque, permet de réduire le temps de développement (un seul patron spécifique à un même problème qui peut être envisagé à la conception et à l'exécution) et de faciliter l'amélioration des patrons, puisque les points faibles des solutions apportées par les patrons sont détectés souvent à l'exécution. De plus certains patrons d'un niveau peuvent collaborer avec d'autres patrons de l'autre niveau. Par exemple, le patron qui est le responsable de l'action consistant à diviser les fenêtres en sous-fenêtres (ce patron n'est utilisé qu'à l'exécution), collabore avec un autre patron (applicable à la conception et à l'exécution) qui est le responsable de l'action consistant à arranger les facettes de présentation.

V.3 Contexte d'usage

Le contexte d'usage du système contient des informations à considérer à la conception et à l'exécution du système (cf. chapitre I, §IV.2.2). Ainsi, le principe de base retenu est le suivant : des informations concernant la plate-forme, l'environnement, et l'utilisateur (P, E, U) sont stockées et actualisées lors de la capture du contexte. Celles-ci sont envoyées au système pour évaluer l'utilisabilité de l'IHM. En cas de changement au niveau du contexte, une adaptation est lancée afin de préserver l'utilisabilité.

V.4 Guide de style

Pendant le cycle de vie de l'IHM, des informations supplémentaires sur la plate-forme cible peuvent être requises, telles que les dimensions maximales autorisées pour les composants de présentation. Le guide de style [Simpson, 1999] contient des règles concrètes qui définissent l'apparence des interfaces pour certaines plateformes [Experience Dynamics, 2007], concernant le style de l'IHM à respecter sur une plate-forme. Le style inclut la structure de l'écran, les couleurs, les polices de texte acceptées, la taille des icônes, etc. Nous avons besoin d'un guide de style afin de s'assurer que l'IHM générée soit cohérente avec le style de plate-forme cible.

La Figure III.15 illustre un extrait du style d'affichage de PDA, et de celui de PC. Nous pouvons distinguer les différences significatives de taille de l'espace d'affichage, des icônes à utiliser dans les messages d'alerte, etc.



Figure III.15 : Exemple de styles de deux plateformes différentes

V.5 Construction de la structure métier

Dans cette étape, la structure métier est construite. Deux grandes catégories de tâches peuvent être distinguées : (1) les tâches mères qui contiennent des sous-tâches (2) et les tâches terminales qui sont des tâches interactives ou automatiques. Nous proposons pour l'instant que la construction de la structure métier soit basée sur huit règles :

1. Un composant métier "conteneur" est associé à un *conteneur abstrait*. La facette de présentation du composant métier peut être une *fenêtre* ou un *widget* lorsqu'une plate-forme graphique est concernée, ou un *menu groupant* si la plate-forme concernée est vocale (une seule touche à appuyer pour aller vers un sous-groupe).

2. Un composant métier "*choix*" est associé à un *conteneur abstrait* qui supervise l'ensemble des *conteneurs abstraits* dont les tâches s'exécutent suite au choix effectué. Le composant métier "*choix*" permet à l'utilisateur d'effectuer un seul choix. Dans ce but, il utilise, par exemple, des composants de présentation de type *RadioButton* pour effectuer le choix sur une plate-forme graphique.
3. En cas de *conteneurs abstraits* dont les tâches associées s'exécutent en concurrence (en parallèle), des composants métier "*panneau*" sont associés aux *conteneurs abstraits*. Ensuite les facettes de présentation des composants métier "*panneau*" sont placées dans la facette de présentation du composant métier associé à leur *conteneur abstrait* mère. Si les *conteneurs abstraits* concernés ont déjà des composants métier "*conteneur*", ces derniers seront supprimés. La facette de présentation du composant métier "*panneau*" peut être, par exemple, un *cadre* pouvant contenir des composants de présentation dans le cas d'une plate-forme graphique, ou un *sous-menu* dans celui d'une plate-forme vocale (une seule touche à appuyer pour aller vers un sous-groupe).
4. Aucun composant métier ne peut être associé à un *conteneur abstrait*, si celui-ci supervise un ensemble de *sous-conteneurs abstraits* ayant déjà des composants métier de type "*conteneur*", et que leurs tâches associées s'exécutent séquentiellement.
5. Des composants métier sont associés aux *composants abstraits*, à l'aide de patrons de conception. La sélection des composants s'effectue en fonction du type de la tâche associée au *composant abstrait*. Les facettes de présentation des composants métier correspondant sont placées dans la facette de présentation du composant métier associé au *conteneur abstrait* mère.
6. Un composant métier "*sortie*" doit être associé aux *conteneurs abstraits* qui ont déjà un composant métier "*conteneur*". Le composant "*sortie*" permet de quitter le système à tout moment.
7. Un composant métier "*validation*" doit être associé aux *conteneurs abstraits* qui ont déjà un composant métier "*conteneur*". Le composant "*validation*" permet de valider les entrées de l'utilisateur et de passer vers la facette de présentation du composant métier associé au prochain *conteneur abstrait/composant abstrait*.
8. Les composants métier sélectionnés, grâce aux précédentes règles, sont reliés entre eux en se basant sur la partie *<relationships>* d'AMXML.

Les règles de construction sont représentées sous forme de patrons grâce à l'outil PaDev, et intégrées dans la bibliothèque des patrons (les patrons sont décrits en annexe D).

Afin de construire la structure métier, tout d'abord, une liste des patrons de conception est choisie (cf. Figure III.16, A2.1.1) pour le choix des composants métier et leur assemblage. Le choix des patrons est effectué selon les tâches et le domaine d'application du système, spécifiés dans l'AMXML ; par exemple, pour des tâches dans le domaine bancaire, des patrons de conception spécifiques à ce domaine sont sélectionnés. Le choix des patrons est basé sur un SIAD (Système Interactif d'Aide à la Décision) [Gachet et Haettenschwiler, 2003 ; Jelassi *et al.*, 1992] considérant les contraintes du domaine et des tâches du système. La connaissance est déterminée par l'expert pendant le développement des patrons de conception. L'étude du système SIAD ne sera pas approfondie dans le cadre de cette recherche.

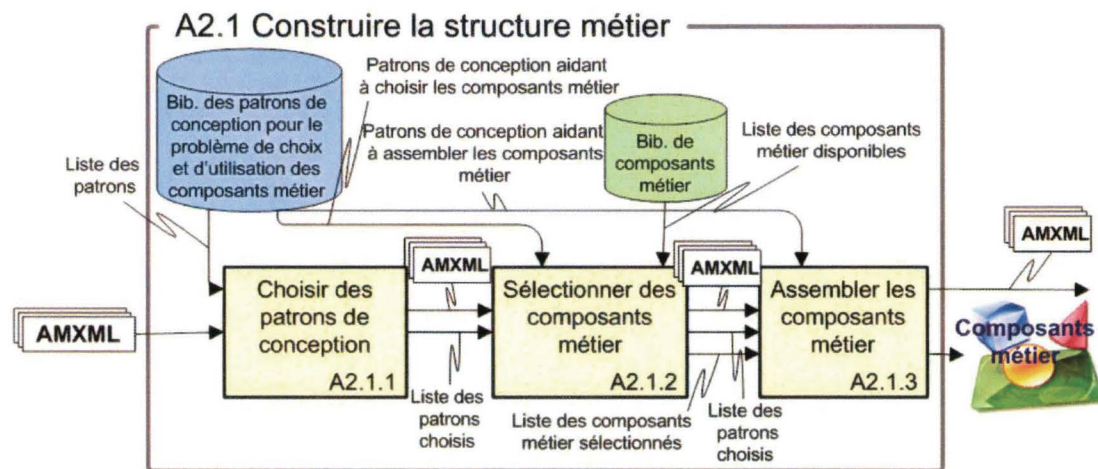


Figure III.16 : Construction de la structure métier

Ensuite, des composants métier sont sélectionnés (cf. Figure III.16, A2.1.2), en respectant les règles ci-dessus, parmi une bibliothèque de composants métier classifiés par domaine d'utilisation (login, recherche, choix, etc.). Les composants métier sélectionnés seront utilisés pour construire la structure métier du système.

Enfin, les composants métier sélectionnés sont configurés et reliés à l'aide des patrons (cf. Figure III.16, A2.1.3). Une « glue » [Beach, 1992], adaptée au *middleware* qui impose une interface de communication, est spécifiée et adaptée au besoin des interfaces de communication des composants métier. Ces interfaces permettent d'accéder aux méthodes et attributs fournis par les composants. Par exemple, le composant métier "recherche", dans le système du serveur de cinéma, qui cherche des films correspondant aux préférences de l'utilisateur, à la fin de la sélection d'un film par l'utilisateur parmi les films trouvés, envoie la sélection par la glue au composant métier "navigation".

Dans notre exemple du serveur de cinéma, une version à base de composants métier est générée (cf. Figure III.17). Trois composants métier (saisie, recherche, et navigation) sont choisis grâce aux patrons de conception et suivant l'AMXML déjà généré. L'interface fonctionnelle (entrées, sorties, relations entre composants) de chaque composant métier choisi est définie selon la partie *links* dans l'AMXML.

Par exemple, les patrons nous orientent automatiquement vers un composant métier utilisable avec la tâche *Recherche* qui s'occupe d'une partie de la tâche mère *Recherche des films par acteur*. Le composant métier "recherche" reçoit la liste des films disponibles par sa propre interface fonctionnelle. Il recherche dans la liste, des films pour un acteur particulier indiqué par l'utilisateur. Puis il affiche le résultat dans sa propre facette de présentation, en particulier dans un composant de présentation *ListBox* ou un autre *DropDownListBox* selon la taille de l'écran et la disponibilité de la modalité graphique. Le film, sélectionné par l'utilisateur, est transféré au composant métier "navigateur" qui permet de passer à la tâche *Réservation*.

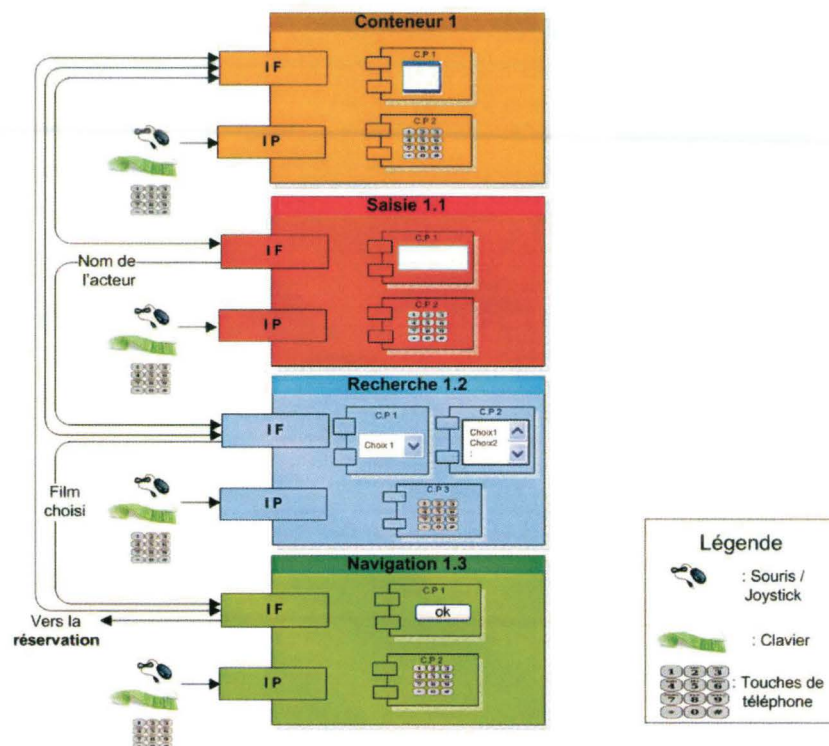


Figure III.17 : Structure métier « recherche par acteur »

V.6 Adaptation primaire

La dernière étape du niveau concret consiste à produire l'IHM finale (cf. Figure III.18). Des patrons de conception sont choisis dans la bibliothèque. Ils seront utilisés pour la sélection des composants de présentation. Rappelons que le composant métier peut posséder plusieurs facettes de présentation utilisables dans différents contextes (voir §IV.1). Les composants de présentation sont encapsulés dans leurs composants métier. Les composants de présentation adéquats au contexte d'usage sont sélectionnés à l'aide des patrons de conception choisis. Ensuite, les composants de présentation sont assemblés et configurés. Les widgets sont arrangés en s'appuyant sur un guide de style qui offre des spécifications et des informations supplémentaires sur le dispositif cible. A ce moment, le code d'une version primaire du système interactif plastique est généré et prêt à être déployé sur la plate-forme.

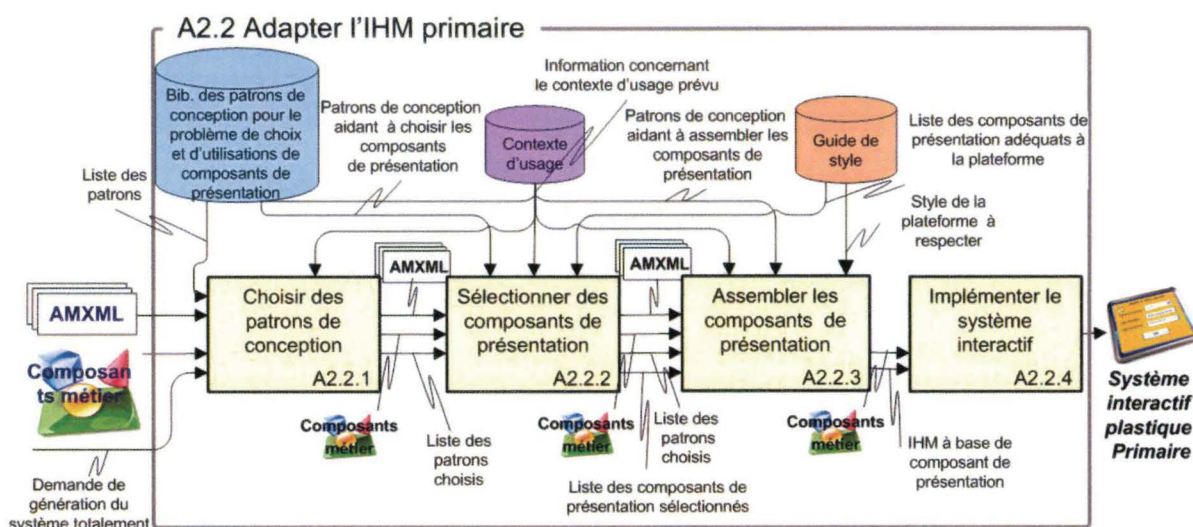


Figure III.18 : Adaptation primaire

Tout d'abord, une gamme de patrons de conception est choisie parmi les patrons stockés dans une bibliothèque (cf. Figure III.18, A2.2.1). Ils sont utilisés pour le choix et l'assemblage des composants de présentation. Le choix des patrons, à cette étape, s'effectue en fonction du type de la plate-forme cible, et des composants de présentation placés dans les facettes de présentation des composants métier. Par exemple, les patrons d'arrangement des facettes de présentation sont différents entre la plate-forme graphique et celle vocale. De plus, le contexte d'usage détermine aussi le type des patrons choisis : par exemple, si un environnement de travail de type bruyant est prévu, des patrons ayant l'objet d'ajouter des messages sonneries peuvent être choisis.

Notons que les problèmes de conception de l'IHM peuvent être différents en fonction du dispositif d'interaction ; par exemple, pour un téléphone portable, les patrons essaient d'utiliser au maximum la capacité de l'écran d'affichage, de plus la plupart des composants sont disposés les un en dessous des autres (verticalement). Alors que sur un PC, les patrons rencontrent le problème de l'harmonisation des facettes de présentation sur les fenêtres, car les patrons ont le choix de placer les facettes de manière horizontale ou verticale).

Ensuite, des composants de présentation sont sélectionnés, en fonction du contexte d'usage, en particulier selon les caractéristiques de la plate-forme (cf. Figure III.18, A2.2.2). La sélection des composants s'effectue grâce aux bases de connaissance des composants métier pour les placer ensuite dans leurs propres facettes de présentation. Les composants de présentation sélectionnés sont ajoutés à l'IHM et configurés en respectant le style de la plate-forme identifiée grâce au contenu du guide de style (cf. Figure III.18, A2.2.3). Nous proposons que quatre règles générales fussent être respectées pendant la génération de l'IHM :

1. Les facettes de présentation des composants métier sont placées dans la facette de présentation du composant métier associé au *conteneur abstrait* mère.

2. Dans le cas d'un opérateur temporel (cf. les opérateurs de CTT, §II.2) de type activation entre deux ou plusieurs tâches terminales (interactives ou automatiques), les facettes de présentation des composants métier correspondant sont activées ou désactivées (affichées/cachées) selon le modèle de tâche.
3. En cas d'opérateur temporel de type activation entre deux ou plusieurs tâches mère, les facettes de présentation des composants métier correspondant sont activées ou désactivées (affichées/cachées) selon le modèle de tâche.
4. La facette de présentation du composant métier "validation" est affichée seulement lorsqu'il y a une saisie ou un choix dans la facette de présentation actuelle du composant métier "conteneur" associé au *conteneur abstrait* mère.

L'harmonisation des composants sur l'IHM prend en compte la chronologie d'exécution des tâches tout en respectant des règles ergonomiques. Dans le cas d'une plate-forme graphique, la taille et le nombre des fenêtres sont indiqués, dans cette étape, par les patrons de conception, et grâce aux informations issues du guide de style. Les patrons considèrent certains critères de la plate-forme (comme la taille de l'écran) et les spécifications de l'espace de travail ; par exemple : l'IHM d'un calendrier peut être vue pour s'étendre sur un espace correspondant à moins de 10% de la zone d'affichage disponible, et souvent dans le coin en bas à droite de l'écran ; mais pour l'IHM de login d'un système bancaire, l'IHM doit s'afficher au milieu de l'écran avec un espace suffisant pour les champs de texte (cf. Figure III.19), afin d'attirer toute l'attention de l'utilisateur.

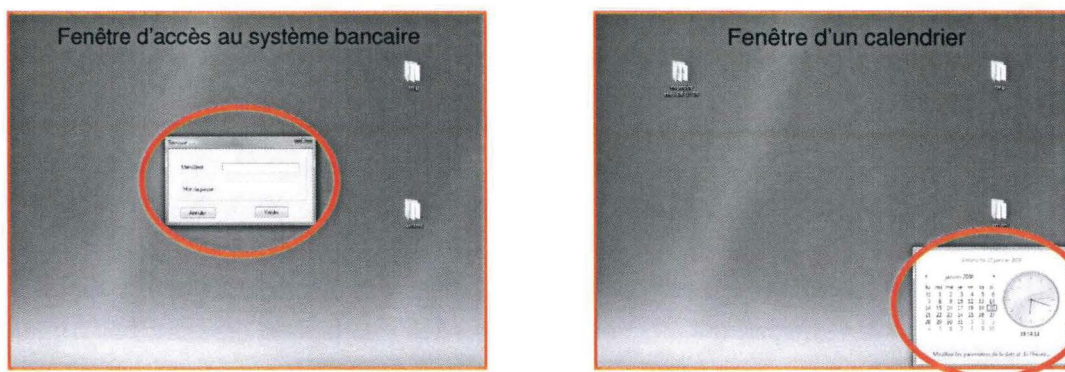


Figure III.19 : Deux localisations différentes pour deux IHM de domaines variés

Enfin, la génération du code suit un ensemble de règles de production (cf. Figure III.18, A2.2.4) qui concernent l'organisation des informations, le style d'interaction, etc., ainsi que les règles de compilation de l'application. L'architecture du système interactif plastique est organisée en trois couches (cf. Figure III.20) :

- La couche de présentation : toutes les facettes de présentation sont accessibles. L'interaction avec l'utilisateur s'effectue dans cette couche.
- La couche fonctionnelle : la structure métier (composants métier assemblés, cf. §IV.5) est située dans cette couche, de même que les processus d'apprentissage et de capture du contexte d'usage. En effet, avant de distribuer le code exécutable sur

le dispositif cible, le système généré est équipé des processus d'apprentissage et de capture du contexte d'usage [Pascoe, 1998]. Par exemple une fonctionnalité de lecture de la luminosité est ajoutée au système, afin de lui permettre de récupérer, pendant l'exécution, le degré de la luminosité de l'environnement¹⁷.

- La couche XML : les bibliothèques de patrons de conception, le guide de style, les règles d'adaptation, et le modèle AMXML sont installés dans cette couche.

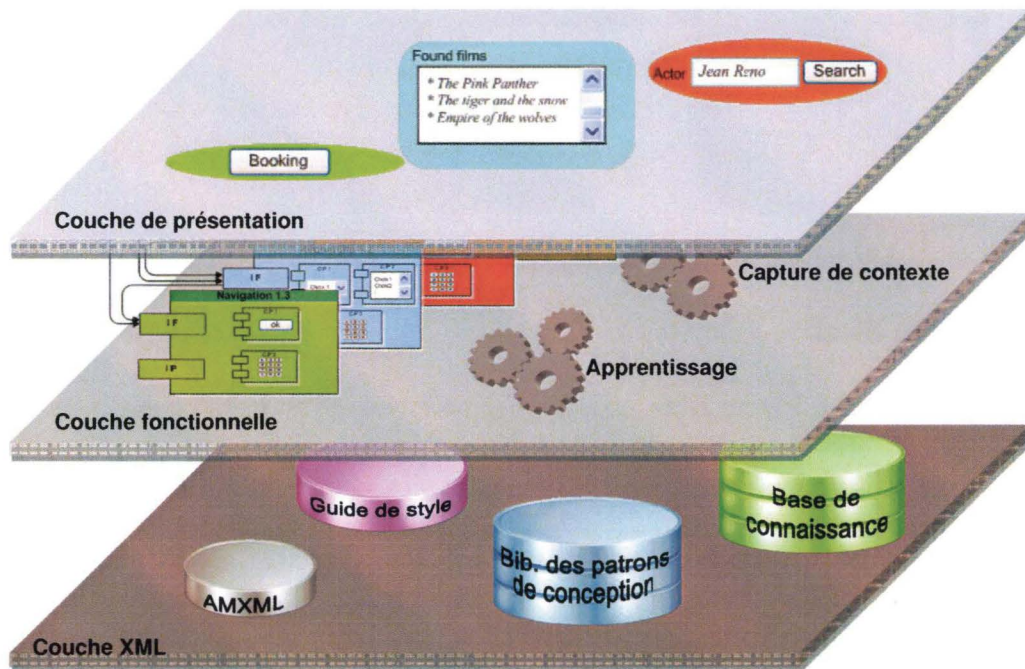


Figure III.20 Architecture du système interactif plastique

Pendant l'adaptation primaire du système de serveur de cinéma, les composants de présentation sont sélectionnés (cf. Figure III.21) selon le contexte d'usage prévu par le concepteur (utilisateur anglais, plate-forme de type PDA, modalité graphique). Par exemple, le composant métier "recherche" choisit un seul composant de présentation de type *ListBox*, parce que l'espace occupé par le composant de présentation de type *ListBox* convient avec la plate-forme de type PDA (taille d'écran moyenne, souris, clavier). Les composants de présentation sélectionnés sont reliés et mis en forme sur l'IHM. Le composant métier "saisie" choisit deux composants de présentation : un de type *EditBox* pour saisir le nom de l'acteur et un autre de type *bouton* pour lancer le processus de recherche. Ensuite, le composant "navigation" choisit un seul composant de type *bouton* pour aller à l'IHM de la tâche *réservation*. Enfin, le code exécutable du système de cinéma est généré et distribué sur le dispositif cible.

Dans la partie suivante, nous allons aborder le niveau final de la méthode en détaillant toutes les activités de ce niveau.

¹⁷ Nous supposons que ces informations proviennent d'un capteur de luminosité sur le dispositif.

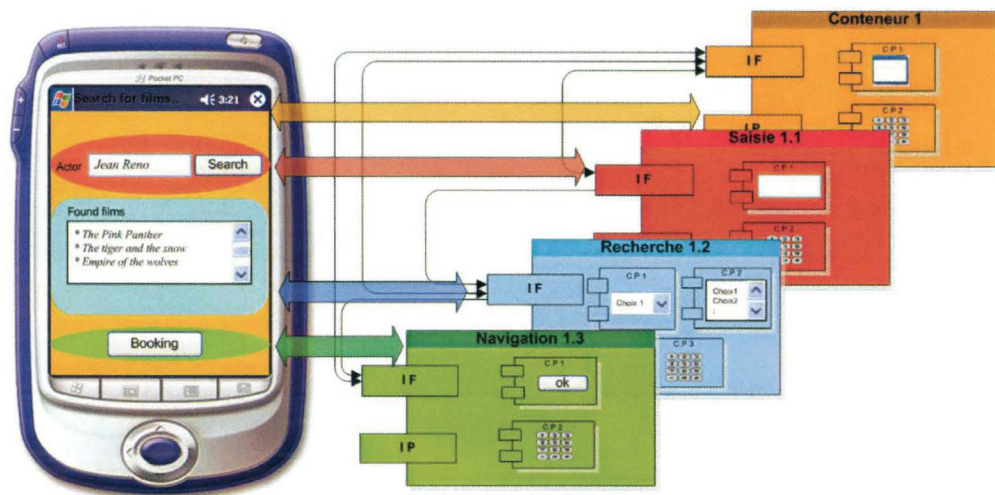


Figure III.21 : Exemple de première version de l'IHM, avec ses composants métier

VI. Niveau final

Au niveau final (cf. Figure III.6, A3), le système interactif est exploité sur la plate-forme cible. Les tâches s'effectuent par l'intermédiaire des composants fonctionnels. Les composants fonctionnels étant déjà à l'intérieur des composants métier, ils forment le noyau fonctionnel du système (l'assemblage et la liaison des composants fonctionnels ne sont pas détaillés dans la thèse). Le contexte d'usage est actualisé avec des informations concernant l'utilisateur, la plate-forme et l'environnement. L'étape d'adaptation vivante est appelée lors d'un changement contextuel. Les processus d'apprentissage travaillent avec le système afin d'améliorer la qualité des décisions à prendre à l'exécution et parfois celles relatives à la conception.

Il existe de nombreux travaux et projets de recherche dans le domaine de l'adaptation de l'IHM, tel que le projet Rainbow (cf. chapitre II, §III.6) qui utilise une infrastructure réutilisable. D'autre part, la majeure partie des travaux, portant sur la plasticité de l'IHM comme [Calvary *et al.*, 2003] et [Thevenin et Coutaz, 1999], prennent en compte normalement une gamme pour l'instant limitée du contexte d'usage comme la taille de l'écran, la langue... Ceci est dû à la difficulté de capturer et de traiter les informations contextuelles. La migration de l'IHM pendant l'exécution, à l'issue de la plasticité, entre les plateformes avec la conservation des données et des informations de l'utilisateur reste un problème difficile. Ce problème de plasticité a notamment été étudié par Paternò et ses collègues [Bandelloni et Paternò, 2004] dans le cadre du projet Caméléon (cf. chapitre II, § III.8). Dans cette section, nous détaillons chaque activité du niveau final, avec des illustrations sur l'exemple du serveur de cinéma.

VI.1 Utilisation du système interactif plastique

Le système interactif plastique offre des mécanismes pour sa manipulation (cf. Figure III.22, A3.1.1) et en parallèle des processus d'apprentissage et de capture de contexte. En cas d'action ou de changement contextuel, l'utilisabilité du système (cf. Figure III.22, A3.1.2) est évaluée afin de provoquer le processus d'adaptation en cas de nécessité. Les

critères d'utilisabilité intègrent la qualité du dialogue qui peut être mesurée par des propriétés du dialogue entre l'utilisateur et l'IHM. Il s'agit d'un ensemble de propriétés centrées utilisateurs. Pendant l'utilisation du système, les réactions de l'utilisateur aux changements de l'IHM, en fonction d'adaptation, sont capturées et envoyées au processus d'apprentissage qui les utilisera pour améliorer les bases de connaissance (abordé en §V.3).

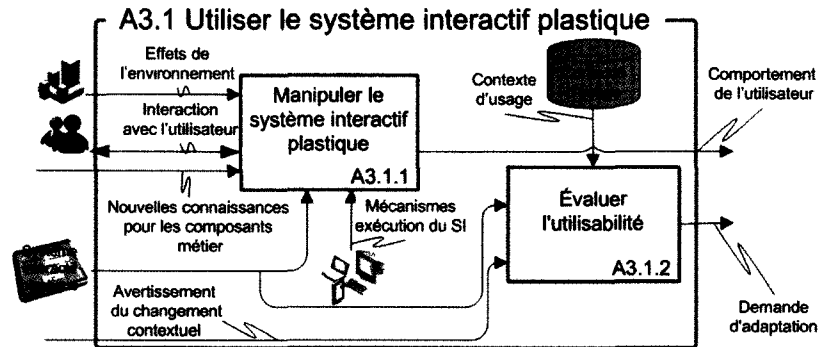


Figure III.22 : Utilisation du système interactif plastique

VI.2 Capture du contexte d'usage

Pendant l'exécution du système interactif plastique, des captures d'informations relatives à l'utilisateur, l'environnement et le dispositif d'interaction sont effectuées (Figure III.23). Dans cette étape, les informations contextuelles détectées [Henricksen et Indulska, 2002, 2004] sont comparées avec les précédentes. En cas de différence, un avertissement sera envoyé à l'IHM, pour lancer l'adaptation vivante, dans le but de conserver l'utilisabilité de l'IHM. Le processus de capture de contexte est lancé automatiquement et de façon autonome en collaboration avec l'adaptation de l'IHM et le processus d'apprentissage pour ré-habiller l'IHM [Kern et Schiele, 2003] en cas d'un changement important au niveau du contexte.

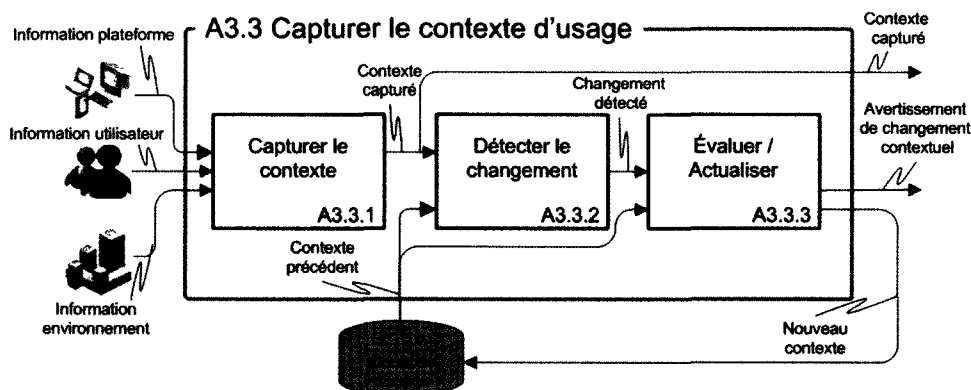


Figure III.23 : Capture du contexte d'usage

VI.3 Processus d'apprentissage

L'apprentissage automatique est un des champs d'étude de l'intelligence artificielle [Cornuéjols *et al.*, 2002] [Russell *et al.*, 2002] [Tsia et Zhang, 2005]. Dans [Kolski et Grislin, 1998], la notion d'interaction intelligente a été présentée relativement à différentes

applications interactives potentielles. L'objet de l'intégration de la notion d'apprentissage dans notre méthode est d'établir un système interactif plastique capable d'engranger des faits et de les organiser selon leur sens ou leurs relations.

Dans notre méthode, la notion de « Action * Condition → Réaction » relative au changement du contexte est ajoutée. Un arbre de décision [Murthy, 1998] est construit pendant la génération de l'interface, et s'installe par la suite dans la couche XML du code distribué (voir Figure III.20). Il est ensuite possible d'améliorer les règles de décision potentielles dans les arbres de décision par le processus d'apprentissage pendant l'exécution. De plus, l'accès à la base de connaissance et les modifications au niveau des règles seront aisément effectués grâce à la représentation de l'arbre de décision en format standard de type XML (un exemple sera proposé par la suite).

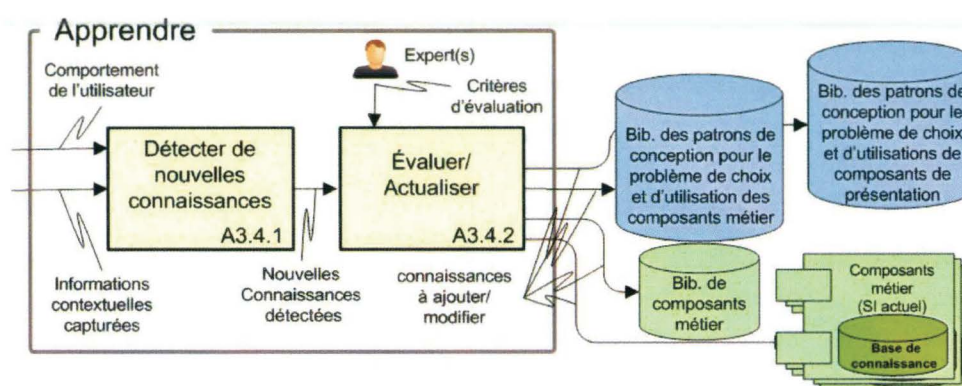


Figure III.24 : Processus d'apprentissage

Le système est basé sur la technique d'apprentissage (Figure III.24) : il apprend à partir d'exemples, au sens des méthodes d'induction [Quinlan, 1979]. Ici, les exemples sont des réactions aux changements du contexte d'usage. Le processus de capture de contexte envoie des informations contextuelles et des informations supplémentaires au sujet des activités et des réactions de l'utilisateur dans certaines conditions. Le système à l'exécution enregistre généralement les réactions de l'utilisateur [Pantic *et al.*, 2006] [Middleton *et al.*, 2001] telles que la modification de la couleur de l'IHM, le changement de la taille d'une fenêtre, ou la réorganisation des facettes de présentation sur l'IHM. Les réactions détectées seront associées aux changements contextuels et aux actions d'utilisation précédemment détectés. De nouvelles connaissances sont détectées en comparant le changement au niveau contextuel et les réactions répétées de l'utilisateur.

Les connaissances extraites sont ensuite évaluées et validées afin d'améliorer et de mettre à jour (1) les bases de connaissances connectées aux composants métier actuels du système (2) celles stockées dans les bibliothèques de composants métier et de patrons de conception. Le système apprend une nouvelle règle qui peut par la suite être appliquée dans des cas semblables, ou il modifie une règle existante déjà dans la base de connaissance [Hilbert et Redmiles, 2000]. Par exemple, une nouvelle règle (cf. Figure III.25) a été ajoutée à la base de connaissance permettant de remettre le fond de l'interface sur un PDA en gris lorsque la luminosité augmente à 75%.


```

<rule> // règle d'adaptation (Luminosité> 75)
  <id>R0000321</id>
  <action>
    <luminosity> // Luminosité est arrivée à plus de 75
      <morethan>75</morethan>
    </luminosity>
  </action>
  <condition> // Plate-forme doit être de type PDA
    <platform>PDA</platform>
  </condition>
  <reaction> // Fond de l'IHM graphique : mis en gris
    <background>
      <color> clGray </color>
    </background>
  </reaction>
</rule>

```

Figure III.25 : Règle d'adaptation

Nous allons maintenant détailler le mécanisme d'apprentissage. Tout d'abord il y a un ensemble de réactions d'adaptation R , et le but de l'apprentissage dans notre cas est de deviner de nouvelles réactions r_j ou améliorer une réaction existante dans la base de connaissance. Notre hypothèse au sujet de la réaction à apprendre est dénotée par h_i qui est choisie parmi une classe des réactions à calculer ou à recalculer H . r_j et h_i sont des réactions pour l'entrée $x \in X$ qui est un sous-ensemble arbitraire d'entrées, $X=A \cup C$ où A est un sous-ensemble d'actions, C est un sous-ensemble de changements contextuels.

Nous proposons que h_i soit implémentée par un dispositif qui a X comme entrée et $h_i(X)$ comme sortie. Il existe plusieurs manières dans lesquelles l'ensemble d'apprentissage S contenant des règles de forme $a.c \rightarrow b$ où $a \in A$ est un sous-ensemble d'actions ; $c \in C$ est un sous-ensemble de changements contextuels, et $b \in B$ est un sous-ensemble de réactions ou de demandes de l'utilisateur.

L'ensemble d'apprentissage S peut être utilisé pour définir ou améliorer une réaction présumée $h_i \in H$. Dans la méthode en lots (*Batch method*) [Osuna *et al.*, 1997], l'ensemble d'apprentissage entier est disponible et utilise tous ses membres en même temps pour calculer une nouvelle réaction h_i . En revanche, dans la méthode par accroissement (*Incremental method*) [Osuna *et al.*, 1997], un membre à la fois est choisi de l'ensemble d'apprentissage S , ensuite il sera seul utilisé pour calculer une réaction h_i , puis un autre membre de l'ensemble d'apprentissage est choisi et ainsi de suite. Le choix des règles peut être aléatoire, ou peut faire itérativement un cycle par les éléments de l'ensemble d'apprentissage S .

$$R = R + \sum_{i=1}^n h_i : h_i \in H$$

Avec : R : ensemble de réactions d'adaptation ;
 h_i : nouvelle réaction calculée.

Nous voyons ci-dessus que l'ensemble de réactions d'adaptation R est mis à jour avec un ensemble de nouvelles réactions calculées h_i .

En revanche, des réactions calculées h_i peuvent remplacer un sous-ensemble de réactions r_j . Dans ce cas, les anciennes sont enlevées de l'ensemble de réactions R, et sont remplacées par de nouvelles réactions calculées h_i . Chaque réaction remplacée r_j et son remplaçant h_i réagit à la même entrée x , sous réserve que l'utilisabilité (Util.) de h_i soit plus grande que celle de r_j .

$$R = R - \sum_{j=1}^{|R|} r_j + \sum_{i=1}^{|H|} h_i : h_i \in H; r_j \in R$$

où : $x_{h_i} = x_{r_j}; Util(h_i) > Util(r_j)$

Avec : R : ensemble de réactions d'adaptation ;
 r_j : réaction remplacée ;
 h_i : nouvelle réaction calculée ;
 Util : fonction d'utilisabilité ;

Par exemple, une réaction possible à une diminution de la taille d'écran (cas de migration de l'IHM), est de réduire la taille de police du texte. Cependant, après avoir réduit la taille de police, il s'avère que l'utilisateur peut essayer de restaurer la taille de police originale, en raison de difficultés à travailler avec la petite police. En outre nous nous apercevons que l'utilisateur peut essayer également d'afficher les barres de défilement afin de s'adapter au problème lié au fait que les composants de l'IHM dépassent la taille de l'espace d'affichage. Par conséquent, la réaction de l'utilisation capturée (barres de défilement) est adoptée comme réaction liée à la diminution de la taille d'écran, au lieu de celle de réduction de la taille de police.

Dans un but d'étude de faisabilité, nous avons adopté l'algorithme C4.5 [Quinlan, 1993] qui est un algorithme de classification utilisé dans le domaine de l'apprentissage supervisé pour construire l'arbre de décision à partir d'exemples. C4.5 utilise le critère de rapport de gain d'informations, et sélectionne le test qui maximise la valeur du rapport de gain plus effectivement que d'autres algorithmes.

Une fois que l'arbre de décision initial est construit, une procédure d'élagage est lancée pour réduire la taille globale de l'arbre, et pour diminuer le taux d'erreurs prévues de l'arbre. La Figure III.26 illustre un arbre de décision simple construit par C4.5 à partir des données, pour la décision *foncer la couleur du texte* pendant l'exécution, selon le niveau de luminosité, en prenant en compte une difficulté visuelle probable chez l'utilisateur. Nous trouvons que l'arbre de décision résume toutes les règles d'une même action mais avec des conditions différentes. Comme les patrons de conception peuvent être appelés au cours de l'application d'une réaction, nous pouvons construire un arbre de décision pour chaque patron à partir des actions et des conditions qui nécessitent l'intervention du patron ; cela permet de spécifier la partie contexte d'usage des patrons (cf. §IV.2.1).

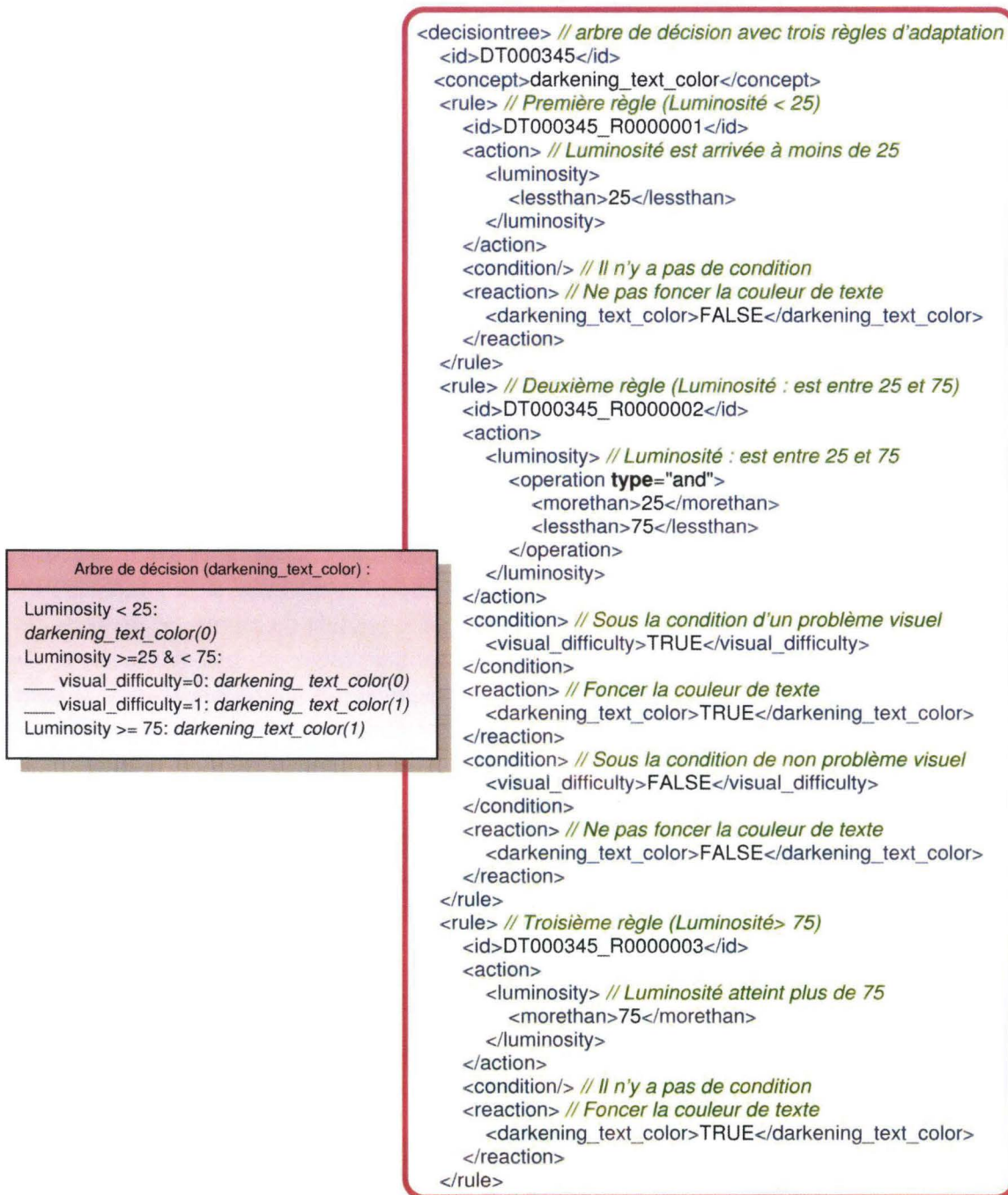


Figure III.26 : Arbre de décision construit avec C4.5 pour la décision de foncer la couleur du texte

Pendant l'implémentation du serveur de cinéma, le processus de capture de contexte commence à envoyer des informations concernant les changements contextuels possibles. L'IHM est équipée de mécanismes donnant la possibilité à l'utilisateur de modifier l'interface (cf. Figure III.27).



Figure III.27 : possibilité de modifier l'IHM à l'exécution par l'utilisateur

Le processus d'apprentissage peut extraire de nouvelles connaissances, et les ajouter à la base de connaissance, ou modifier des règles existantes. Par exemple si la luminosité de l'environnement atteint un certain niveau, capturé par le module de capture de contexte, le système s'adapte à ce changement en appliquant une règle choisie de la base de connaissance (la règle indique que : *si la luminosité atteint un haut niveau, alors le fond sera remis en gris clair*). Suite à l'application de cette règle, les informations capturées du côté de l'utilisateur indiquent que celui-ci avait répété une même réaction, à la réaction automatique précédente du système, en refusant directement la couleur grise et en modifiant tout de suite la couleur du fond à bleu clair au lieu de gris. Par conséquent le processus d'apprentissage déduit cette réaction de l'utilisateur, puis la couleur adoptée dans la règle appliquée dans la condition de la luminosité élevée, sera remplacée par celle bleu clair, parce que dans notre étude de faisabilité, notre hypothèse est que cette couleur est plus appropriée à l'utilisateur actuel (Figure III.28). La modification de la décision est effectuée soit par l'algorithme *C4.5* en prenant en compte les répétitions au niveau des exemples en entrée de l'algorithme, ou soit à tout moment, par la possibilité d'accéder à la base de connaissance et modifier directement la décision.

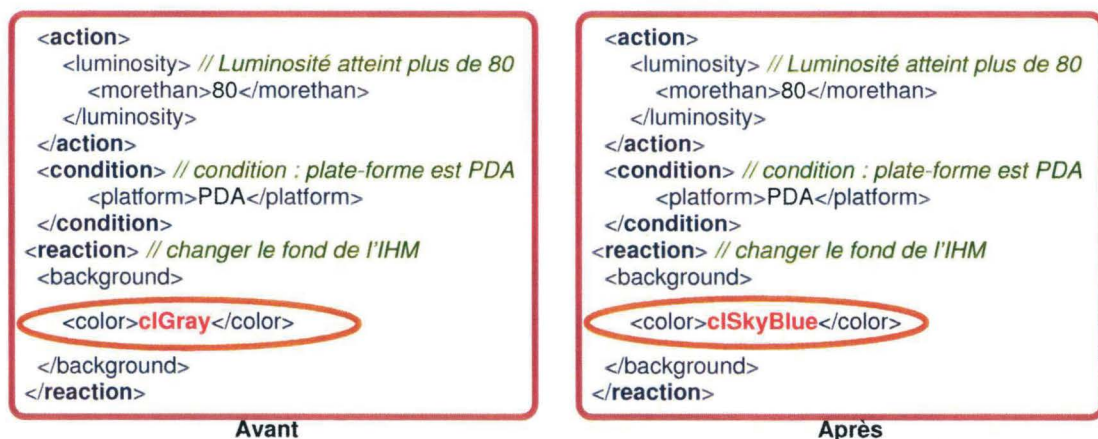


Figure III.28 : Base de connaissance avant et après le processus d'apprentissage

VI.4 Adaptation vivante

Dans le cas où l'adaptation est nécessaire durant l'exécution (cf. Figure III.29), l'IHM est évaluée selon les critères d'utilisabilité pour décider le type d'adaptation nécessaire. Cette évaluation permet de savoir si l'adaptation vivante est capable de prendre en compte les changements contextuels. Si tel est le cas, une ou plusieurs parties de l'IHM qui doivent être modifiées sont détectées afin de préserver l'utilisabilité de l'IHM. Puis une liste des patrons de conception relatifs à la sélection des composants de présentation est choisie. Les composants de présentation sont re-sélectionnés à l'aide des bases de connaissance de leurs composants métier, adéquatement au nouveau contexte d'usage. Les composants de présentation sont réassemblés ; le code implémentable de la partie adaptée est mis à jour. Par contre si l'interface n'est plus capable de s'adapter, le système désassemble les composants de présentation de certains composants métier. Ces composants métier sont transmis à l'adaptation primaire du niveau conception afin de régénérer totalement l'IHM.

Nous avons implémenté une application directe de l'adaptation vivante (cf. annexe E) qui nous permet de montrer l'intérêt ainsi que la nécessité de l'adaptation vivante, prenant la forme d'un démonstrateur.

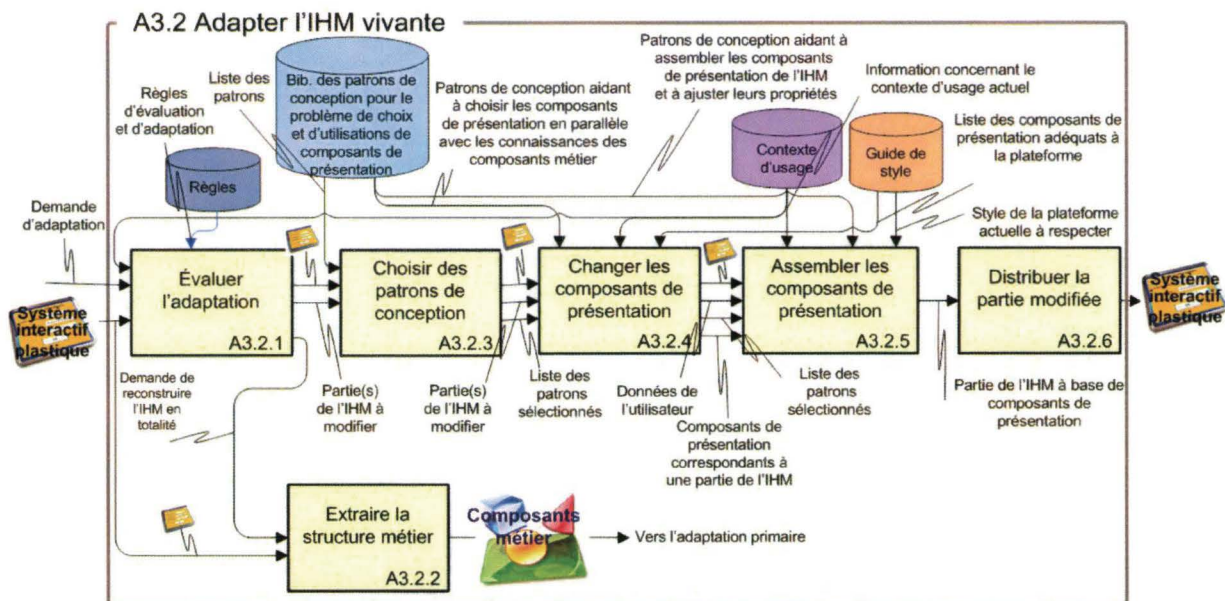


Figure III.29 : Processus d'adaptation vivante

Pendant l'exécution du serveur de cinéma, l'IHM reçoit, par exemple, une demande de migration vers une autre plate-forme. Par conséquent, l'IHM est transférée au processus d'adaptation vivante, car l'écran d'affichage de la nouvelle plate-forme est deux fois plus petit que le dernier écran. Tout d'abord, le composant métier *recherche* change sa facette de présentation en choisissant le composant de présentation *DropDownListBox* au lieu de celui de type *ListBox*. Le patron de conception P10201 (cf. Figure III.13) est choisi pour régler la taille et les localisations de la fenêtre et de ses composants. Grâce au gabarit adopté par notre méthode qui archive des informations sur chaque patron afin de faciliter le choix du patron adapté à un problème, dans notre cas le problème est la taille d'écran

réduite. Ce patron appelle un autre patron P10226 (diviseur de fenêtres) qui applique des règles de division sur la fenêtre en raison du taux significatif de changement de taille de l'écran. Le patron diviseur calcule l'espace nécessaire pour montrer les composants de présentation correspondant à l'IHM qui est en cours d'adaptation. Ensuite une décision est prise pour diviser les fenêtres originales horizontalement et/ou verticalement en sous-fenêtres adéquates à l'espace d'affichage disponible. Enfin le dernier patron P10201 est rappelé pour réarranger les composants de présentation de chaque sous-fenêtre.

En conséquence, la fenêtre *Recherche des films par acteur* est devenue transformée en deux petites fenêtres (cf. Figure III.30) adaptées au nouvel espace d'affichage. Cette transformation a permis de préserver l'utilisabilité de l'IHM au nouveau contexte¹⁸.



Figure III.30 : L'IHM de la tâche « recherche par acteur » réadaptée selon la taille d'écran du téléphone mobile

VII. Conclusion

Les travaux de recherche présentés dans ce chapitre ont visé à proposer une méthode de génération d'IHM plastiques. La méthode se base sur l'utilisation de patrons permettant de faire des choix concernant les composants métier relatifs aux tâches du système et de configurer, d'assembler et de relier les composants de présentation de manière à ce qu'ils soient adaptés au contexte d'usage. Comme les principes d'adaptation sont basés sur des décisions déjà définies dans des bases de connaissance, nous avons intégré la notion d'apprentissage dans la méthode, afin d'améliorer la qualité des décisions à différents

¹⁸ Notons que [Findlater et McGrenere, 2008] ont étudié en profondeur la migration de l'IHM d'un espace d'affichage à un autre plus petit (comme dans notre dernier cas) ; ils ont mesuré la satisfaction et la performance de l'utilisateur avant et après la migration de l'IHM. Ils ont accompli leur étude sur 36 sujets afin de comparer l'IHM dans les deux espaces d'affichage. Les résultats montrent que les menus adaptés de l'IHM ont un impact important sur la performance et la satisfaction dans le cas d'un espace d'affichage limité ; par contre l'inconvénient de ces menus adaptés est de réduire potentiellement la visibilité de l'ensemble de l'IHM.

niveaux. Nous avons montré plusieurs exemples sur l'évolution des règles d'adaptation suivant le comportement de l'utilisateur face à des changements contextuels. Une DTD a été proposée basée sur UsiXML. La DTD est utilisée pour décrire notre modèle AMXML. Le MIA de notre méthode (AMXML), les patrons de conception, et les règles d'adaptation sont décrits en XML ; cela permettant d'accéder à ceux-ci et de les modifier aisément. Nous avons présenté les différentes étapes de la méthode avec les activités détaillées de chaque niveau (en se basant sur les étapes du projet Cameleon) à partir de la représentation du MIA original en AMXML suivant la DTD proposée et en finissant avec l'IHM finale. Le cycle de vie de l'IHM plastique est considérablement différent de celui de l'interface classique : la différence majeure se situe à l'exécution où les interfaces plastiques essaient de gagner en temps et coût de maintenance qui seraient perdus progressivement en cas d'évolution des systèmes, dans le cas de l'interface classique. Nous avons donc essayé de nous concentrer sur la troisième étape du cycle de vie de l'IHM (à l'exécution).

Un exemple de serveur de cinéma a accompagné toutes les étapes de la méthode. Le but de l'exemple était d'éclaircir certaines idées difficiles à comprendre sur le modèle IDEF0. Dans le chapitre suivant, une première évaluation de notre méthode est fournie en s'appuyant sur deux exemples concrets : la première étude de cas concerne un système applicable dans la vie quotidienne, permettant de guider des touristes dans une ville tout en leur proposant des services ; une seconde étude provient du monde industriel, puisqu'il s'agit d'un procédé industriel de fabrication de godets métalliques remplis par une solution chimique préparée préalablement.

Chapitre IV

**Simulation d'implémentation de la démarche
sur deux cas d'étude**

I. Introduction

Nous avons proposé dans le chapitre précédent une démarche de génération d'IHM plastique à partir d'un modèle d'interface abstraite et/ou un modèle de tâche. Cette démarche est basée sur les composants métier. Les composants métier ont la capacité de sélectionner leur facette de présentation (un ou plusieurs composants de présentation) et de la changer à l'exécution. Les patrons de conception sont classifiés en deux catégories selon leur utilisation (sélection et assemblage des composants métier, et sélection assemblage, et arrangements des composants de présentation). Nous allons dans ce chapitre valider et implémenter notre démarche en l'appliquant sur deux études de cas de système interactif plastique, destinés à deux environnements d'utilisation différents. Le premier est un système de guidage touristique qui permet d'indiquer le chemin à suivre aux visiteurs et leur délivre des informations sur la visite. Les touristes peuvent, par exemple, utiliser ce système afin de trouver un restaurant, un hôtel proche d'eux, ou réserver des billets de cinéma (proposé au chapitre III), etc. Le deuxième exemple est un système de supervision d'une usine chimique. Ce système surveille et contrôle les données de conduite de l'installation (température, humidité, alimentation électrique, stocks, fonctionnement des machines...). Ce système prévient les utilisateurs en cas d'alerte ou d'incidents afin d'intervenir.

L'illustration de la méthode en deux cas d'étude différents permettra de distinguer les points forts et les points faibles de la méthode. Chacun des deux exemples a ses propres caractéristiques, ses contextes d'usage spécifiques, et sa fréquence de changement du contexte.

II. Premier cas d'étude : système de guidage touristique

Nous commencerons par présenter le scénario du système de guidage de touriste ; ensuite le contexte d'usage envisagé sera détaillé. Enfin le système sera généré en suivant les étapes de la méthode proposée.

II.1 Présentation de l'application

On suppose que la mairie d'une ville touristique décide de mettre à disposition des visiteurs un système de guidage touristique. Ce système offre la possibilité de choisir le type de visite ; touristique, shopping, travail, etc. Pendant la visite, le système propose aux touristes plusieurs choix de parcours de visite, indique le chemin à suivre, et délivre des informations sur les points d'intérêt proches du visiteur. Tout au long des parcours, le système peut délivrer au touriste, toutes sortes d'informations sur les caractéristiques d'un lieu touristique, ou encore sur la promotion d'une gamme de vêtements en passant devant un magasin.

Les touristes, dans cette ville, peuvent utiliser ce système pour trouver un endroit tel qu'un restaurant ou un hôtel à proximité, pour obtenir des informations sur un endroit (une place, une rue, un bâtiment, un monument...), pour connaître les itinéraires de visite, etc. Le

système de guidage touristique assure une information continue sur les événements, les fêtes, et les festivals ayant lieu dans la ville. Le système peut aussi chercher des séances de cinéma et réserver des billets pour les touristes. Le système sera exécuté sur les terminaux des visiteurs de la ville (PC portable, PDA, téléphone cellulaire...), ensuite l'IHM s'adaptera au contexte d'usage, par exemple le type du dispositif utilisé par le touriste, sa langue et son âge, le but de la visite, le temps, la luminosité de l'environnement, le niveau de bruit, etc.

Pour donner une idée du guide de visite, il est possible de faire l'analogie avec le système "Geovisite"¹⁹ proposé par France Telecom (cf. Figure IV.31). Ce système permet de réaliser des guides interactifs proposant aux visiteurs de sites touristiques, des informations enrichies en multimédia, etc., en utilisant la technique de communication Wifi sur les PDA. Cependant, ce système ne supporte pas la propriété de plasticité.



Figure IV.31 : Guide de visite "Geovisite" proposé par France Telecom

II.2 Modèle de tâche et Modèle d'interface abstraite

Les tâches du système de guidage touristique sont exprimées dans un modèle de tâche s'appuyant sur CTT (cf. Figure IV.32). Le système se compose de cinq tâches principales :

- *Proposer* : le système fait des propositions au touriste comme par exemple, aller au théâtre, au cinéma, ou visiter un musée de la ville.
- *Chercher* : l'utilisateur peut chercher une banque, un centre commercial, la gare ou des lieux intéressants (par exemple, piscine, restaurant,...).
- *Gérer agenda de visite* : l'utilisateur peut planifier sa visite dans cette partie, comme par exemple, planifier une visite au musée, ou une participation à un événement, etc.

¹⁹ Il est accessible sur le site :

<http://www.francetelecom.com/fr/groupe/rd/offre/logiciels/technologies/middlewares/Geovisite.html>

- *Fournir informations* : le système offre au client des informations concernant les éventuels événements spectacles, festivals..., les informations météorologiques, ou encore les informations trafics.
- *Trouver chemin* : le système permet de guider l'utilisateur vers une adresse ou un lieu déjà spécifié, ou d'explorer la carte de la ville.

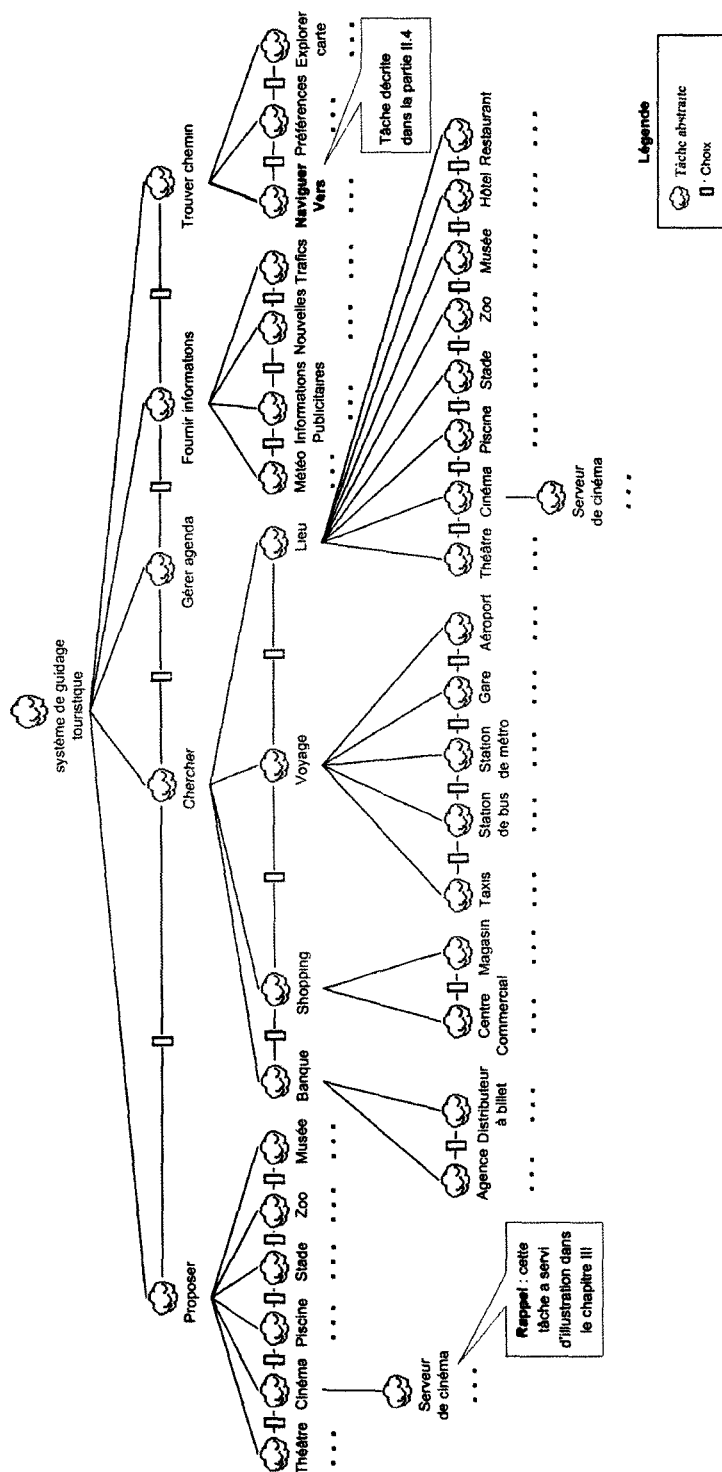


Figure IV.32 : Modèle de tâche du système de guidage touristique représenté en CTT

Nous partons de l'hypothèse que le modèle d'interface abstraite est spécifié en UsiXML. Des *conteneurs abstraits* sont associés à la tâche racine, et aux sous-tâches abstraites. Ils sont nommés de la même manière que les tâches leur correspondant. Les conteneurs sont organisés en respectant la hiérarchie des tâches. La Figure IV.33 illustre le premier niveau du modèle abstrait incluant un *conteneur abstrait* *Système de guidage touristique* correspondant à la tâche racine et cinq *sous-conteneurs abstraits* correspondants aux sous-tâches de la tâche racine : *Proposer*, *Chercher*, *Gérer agenda*, *Fournir informations*, et *Trouver chemin*.

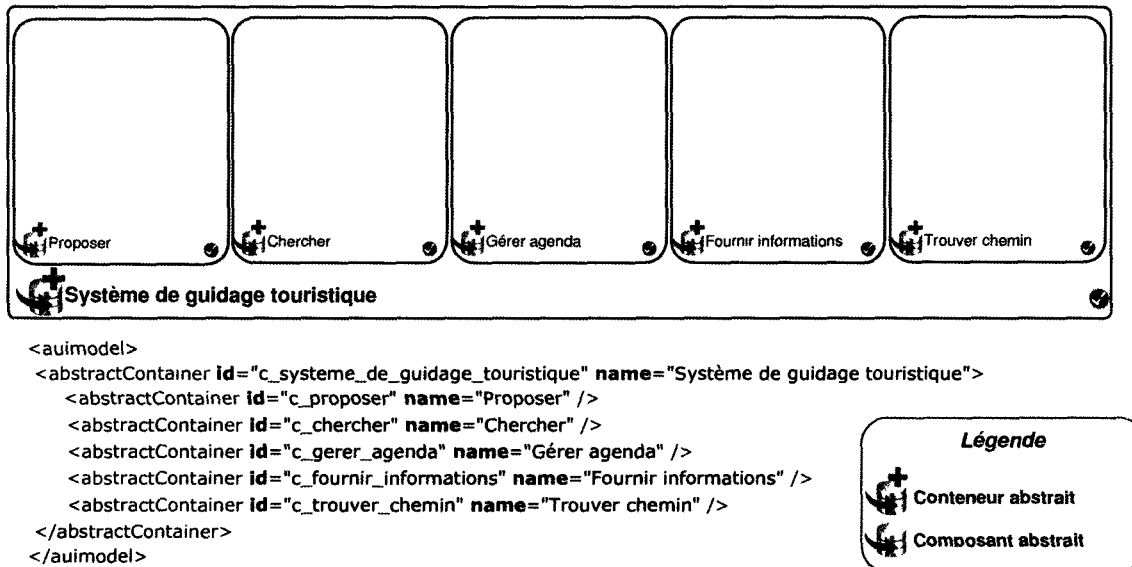


Figure IV.33 : Modèle d'interface abstraite de la tâche racine en UsiXM
(à l'aide de l'outil IdealXML cf. Chapitre II, § III.9.2)

II.3 Contextes d'usage envisagés

Différents contextes d'usage (P : plate-forme, U : utilisateur, et E : environnement) sont envisagés pour le système interactif plastique ; rappelons que ces contextes doivent être considérés afin de préserver l'utilisabilité du système.

II.3.1 Contexte relatif à l'utilisateur

Nous détaillons dans cette partie, les différentes facettes du contexte relatif à l'utilisateur (cf. Tableau IV.3). Dans un but d'étude de faisabilité, nous supposons que le touriste qui utilise ce système de guidage peut être un enfant ou un adulte.

Le système de guidage peut prendre en compte la profession du visiteur, afin de lui fournir des informations qui pourraient l'intéresser ; par exemple, pendant la visite d'un restaurant aménagé dans un bâtiment ancien, si le visiteur est architecte, le système lui fournit des

informations architecturales en lien avec ce restaurant. Ces informations peuvent être émises par des étiquettes communicantes²⁰ installées dans le restaurant.

Concernant les aspects d'internationalisation du système, l'IHM doit avoir la possibilité d'afficher les textes et messages dans la langue de l'utilisateur. De plus, pour un visiteur possédant un handicap, l'IHM devra par exemple utiliser des couleurs spéciales pour le fond et pour les composants de présentation, des messages sonores et/ou des alertes vibrantes selon le type de handicap²¹.

Le système peut prendre en compte le but et la durée de la visite, afin de filtrer les informations qui peuvent être proposées au visiteur. Par exemple, si le visiteur vient en ville dans le but de faire du shopping, l'IHM peut lui proposer des informations émises lors du passage devant les magasins.

Enfin, l'IHM doit respecter les préférences de l'utilisateur en ce qui concerne les couleurs (fond d'écran, texte,...) et les modes d'interaction préférés de l'utilisateur si plusieurs outils d'interaction sont disponibles sur la plate-forme cible (clavier, souris, joystick, ...).

Tableau IV.3 : Facettes du contexte relatif à l'utilisateur

Âge	Profession	Expérience en informatique	Langue	But de la visite	Durée de la visite	Mode d'interaction préféré	Couleurs préférées	handicap
Enfant	Architecte	Aucune	En	Shopping	Quelques heures	Vocal	Noir/blanc	Aucun
Adulte	Enseignant	Basse	Fr	Business	Une journée	Tactile	Bleu/blanc	Visuel
	...	Moyen	De	Recherche	Deux jours	Clavier	...	Auditif
	...	Avancé	Ar	Joystick

II.3.2 Contexte relatif à l'environnement

Les caractéristiques de l'environnement en lien avec la ville, où le système de guidage sera exploité, sont variées et fréquemment changées. L'IHM doit prendre en compte une gamme de facettes du contexte relatif à l'environnement (cf. Tableau IV.4). Comme l'utilisateur sera probablement en situation de mobilité, les caractéristiques de l'environnement sont instables, et le système de guidage doit s'adapter à ces changements.

²⁰ Etiquette communicante : est une puce électronique qui s'intègre dans un système d'information et interagit avec son environnement. Elle est capable d'émettre des informations aux dispositifs à proximité ou de capturer des informations contextuelles [www.infopole.be].

²¹ Rappelons que cet aspect fait l'objet d'un courant de recherche au niveau international, relatif à l'accessibilité.

Le système peut être utilisé dans un environnement fermé (par exemple à l'intérieur des bâtiments) où le niveau de luminosité peut être relativement stable, ou à l'extérieur (par exemple dans une rue) où celui-ci changera en fonction du climat (ensoleillé, éclaircies, très nuageux,...) et de l'heure (le système peut être utilisé pendant la journée, avec un haut niveau probable de luminosité, ou pendant la nuit avec un niveau de luminosité très bas).




L'utilisation du système peut se faire dans un environnement bruyant (ex. dans la rue ou dans un hypermarché) ou calme (ex. café peu fréquenté, musée ou cinéma). Dans un environnement bruyant, il sera éventuellement nécessaire d'utiliser un niveau sonore élevé couplé à des alertes vibrantes. En revanche, dans un lieu calme, l'IHM devrait arrêter l'utilisation des messages sonores, et n'utiliser que des alertes vibrantes.

Le climat peut jouer un rôle essentiel dans le système de guidage ; par exemple, selon le climat, le système peut offrir aux touristes des conseils ou des propositions concernant leur visite. Par exemple, il ne sera pas évident de proposer la visite d'un jardin, s'il pleut toute la journée.

Une possibilité de détection de la position de l'utilisateur peut être utilisée (en utilisant pour l'instant le système de GPS ou le réseau ad-hoc). Avec cette possibilité, l'IHM peut fournir aux visiteurs des informations utiles ou une assistance automatique en fonction de leur position. Par exemple, le système de guidage peut trouver le positionnement des visiteurs proches et les contacter en cas de nécessité ou d'urgence ; par exemple, si un enfant visite la ville avec sa famille et que soudainement il perd ses proches, le système devra pouvoir utiliser son système de localisation afin de retrouver ses parents et/ou les contacter.

L'adaptation de l'IHM à l'environnement dépend plus de l'adaptation vivante (lors de l'exécution, cf. chapitre III § V.4) que de l'adaptation primaire (au cours de la conception, cf. chapitre III § IV.6), puisque les caractéristiques de l'environnement changent vite.

Tableau IV.4 : Facettes du contexte relatif à l'environnement

Lieu	Mobilité	Luminosité	Bruit	Heure	Etat	Météo	Température
Restaurant	Aucune	Basse	Bruyant		Urgent		
Rue	Basse	Moyenne	Calme	Valeur au format heure (ex : 12:00)	Pas d'alerte		Valeur numérique (ex : 35°)
...	Moyenne	élevée			Pas d'information		
	Elevée				

II.3.3 Contexte relatif à la plate-forme

Le système est utilisé sur les plateformes les plus connues telles que le PC, le PDA, le téléphone cellulaire, etc. Puis l'IHM devrait aussi pouvoir s'adapter à une plateforme cible inconnue (voir chapitre I, § IV.2.3). Le processus d'adaptation doit prendre en compte les différentes caractéristiques de la plate-forme cible (cf. Tableau IV.5) :

- Type : il précise le type de la machine, par exemple un PDA, ou un téléphone portable ; selon le type de la machine, on peut spécifier des tâches qui ne peuvent pas s'exécuter sur celui-ci, et vice versa. Par exemple, le concepteur peut spécifier, dans le modèle de tâche, plusieurs tâches de même objectif (par exemple, recherche) pour des plateformes différentes (l'une pouvant être exécutée sur un PDA, une autre sur un PC...). Cela permet de prendre en compte la puissance de traitement de la machine.
- Système d'exploitation : le style d'affichage ou de présentation de l'IHM peut être varié en fonction du système d'exploitation de la machine (par exemple, Linux et Windows peuvent être installés sur un PC, mais ils n'ont pas le même style d'affichage). Cela permet au processus d'adaptation de choisir et de configurer les composants de présentation en accord avec le guide de style.
- Espace d'affichage : il permet de spécifier l'espace d'affichage disponible pour l'IHM et pour chaque composant (cas d'une plate-forme graphique). L'espace disponible doit être respecté aussi, dans la mesure du possible (l'IHM ne doit pas dépasser l'espace disponible) et exploité (on doit procéder à un arrangement des composants de présentation dans cet espace). Dans le cas d'une plate-forme vocale, il y a des temps de traitement ou de réponse à respecter par l'IHM ; par exemple, le temps d'écoute des choix proposés par le système pour réserver un billet de cinéma, auquel s'ajoute le temps nécessaire à l'utilisateur pour répondre, ne doit pas dépasser un nombre limité de minutes.
- Connexion réseau : la prise en compte de la disponibilité d'une connexion, de la vitesse de la connexion et des protocoles de connexion sont indispensables à l'exécution. La connexion réseau peut être utilisée pour mettre à jour la base de données du système, la base de connaissance, les images et les icônes, la réservation (par exemple, la réservation d'un billet de théâtre).
- Système de localisation : afin de guider l'utilisateur pendant sa visite, un système de localisation est indispensable pour le système de guidage. En cas d'indisponibilité du système de localisation, le système doit pouvoir passer à la tâche *explorer carte* (voir Figure IV.32). Grâce à cette tâche, l'utilisateur peut savoir la direction et le chemin à suivre lui-même.
- Couleurs : les couleurs disponibles sur une plate-forme graphique permettent de spécifier les images qui peuvent être utilisées dans l'IHM (résolution de l'image), ainsi que les couleurs des composants de présentation.
- Modes d'interaction disponibles, comme par exemple : clavier, clavier avec joystick, écran tactile, etc.
- Style : sur la même machine avec un même système d'exploitation, nous pouvons envisager des styles d'affichage différents (par exemple, sur un PC exploité par linux, nous avons plusieurs styles d'affichage comme Gnome, Kde...).

Tableau IV.5 : Facettes du contexte relatif à la plate-forme

Type	Système d'exploitation	Connexion réseau	Système de localisation	Mode d'affichage	Espace d'affichage	Couleurs	Modes d'interaction disponibles	Style
PDA	Windows mobile	Non	Non	Horizontal	220*300	4096	Clavier	Fenêtre, Menu, ...
Téléphone cellulaire	Symbian	Wifi	Ad-hoc	Vertical	640*480	65536	Clavier/ Joystick	Page Web, bouton, ...
...	Linux	...	GPS	Ecran tactile	Vocal
	

II.4 Application de la démarche

Nous allons dans cette partie mettre en application notre démarche, proposée dans le troisième chapitre, sur le système de guidage touristique. Nous partons d'un modèle de tâche du système spécifié en CTT. L'AMXML peut être construit en appliquant des règles de construction d'AMXML proposées dans le chapitre III. Ensuite, la structure métier sera générée. Une version primaire du système sera générée et sera prête à être distribuée vers le dispositif cible. La version primaire du système peut être différente en fonction du contexte d'usage prévu. L'adaptation vivante sera appelée à l'exécution lors d'un changement contextuel considérable (lors de l'évaluation de l'utilisabilité de l'IHM).

II.4.1 Construction d'AMXML

Les tâches du système de guidage touristique sont exprimées selon la modélisation CTT (cf. Figure IV.32). Comme le système de guidage touristique est vaste, nous nous intéresserons seulement à la tâche *naviguer vers* (cf. Figure IV.34).

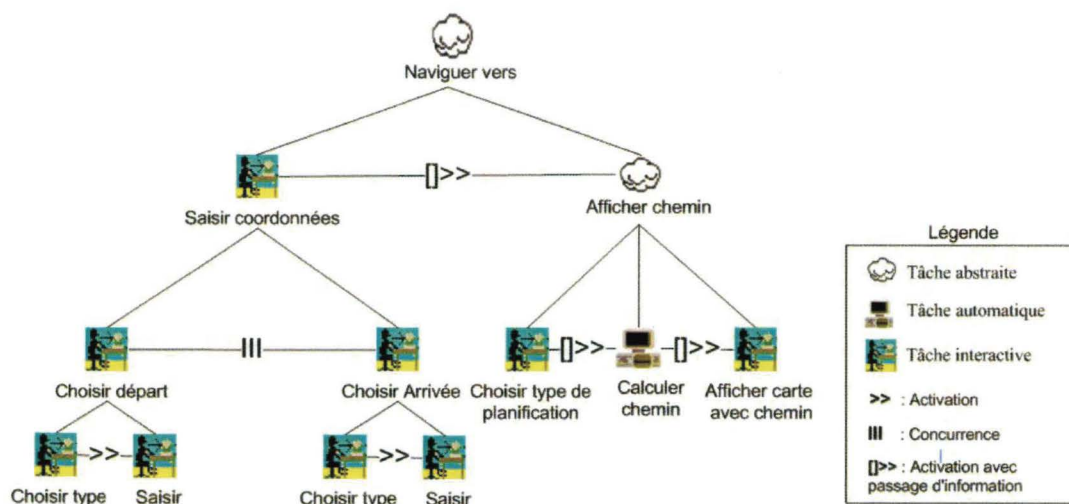


Figure IV.34 : Partie du modèle de tâche concernant la sous-tâche « Naviguer vers »

Afin d'effectuer la tâche *Naviguer vers*, les coordonnées du départ et celles de l'arrivée sont nécessaires. L'utilisateur peut saisir directement l'adresse complète de départ et celle

d'arrivée, le nom d'un lieu (hôtel, restaurant, ...), ou le nom d'une station de transport à proximité. Ensuite, selon les informations saisies, le chemin est calculé en fonction de type de planification choisi par l'utilisateur (itinéraire pédestre, itinéraire cycliste, ...). Ensuite le chemin à suivre est affiché sur la carte de la ville.

Afin de générer le modèle AMXML de notre méthode, les règles de génération d'AMXML (voir chapitre 3 §III.3) seront appliquées, au fur et à mesure, sur le modèle de tâche ci-dessus. Tout d'abord, nous appliquons la règle 2 sur le modèle de tâche : par conséquent, des *conteneurs abstraits* sont associés à la première tâche *Naviguer vers*, et aux sous-tâches mères *Saisir coordonnées*, *Afficher chemin*, *Choisir départ*, et *Choisir arrivée*. Ensuite nous appliquons la règle 3 qui permet d'associer des *composants abstraits* aux tâches terminales *Choisir type* et *Saisir* de la tâche mère *Choisir départ* et celles de la tâche mère *Choisir arrivée*, ainsi que les tâches terminales *Choisir type de planification*, *Calculer chemin* et *Afficher carte avec chemin* de la tâche mère *Afficher chemin*. Les *composants abstraits* sont placés dans les *conteneurs abstraits* de leurs tâches mères en respectant la règle 4. La Figure IV.35 illustre les *conteneurs abstraits* et les *composants abstraits* associés aux tâches et représentés en UsiXML. Les *conteneurs abstraits* et les *composants abstraits* sont organisés en respectant la hiérarchie des tâches du modèle en question.

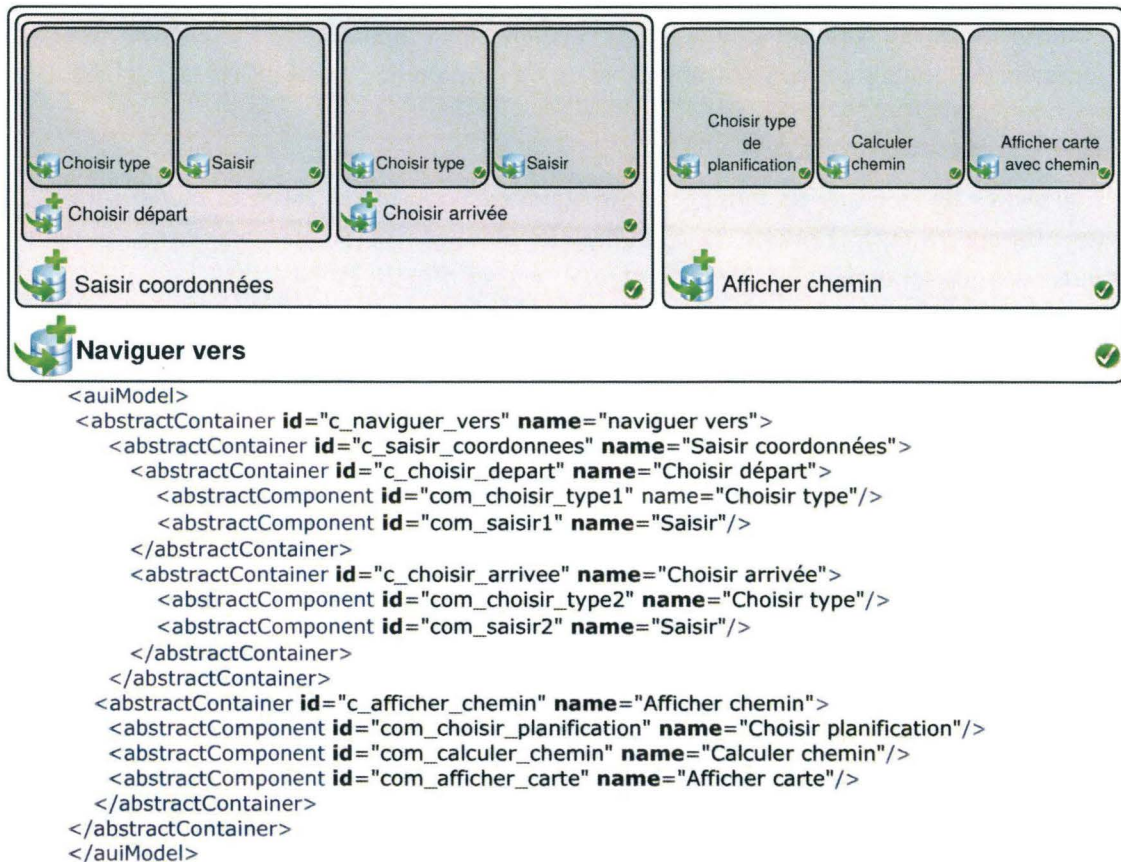


Figure IV.35 : Modèle d'interface abstraite de la tâche « Naviguer vers » en UsiXML

La construction du modèle AMXML se poursuit en complétant la partie *<taskModel>*, des balises *<task>* sont associées à chaque tâche dans le modèle de tâche. Ensuite les opérateurs entre les tâches (activation, concurrence,...) sont intégrés dans la partie *<relationships>*. Une balise *<link>* est associée à chaque opérateur, en spécifiant les deux tâches concernées par l'opérateur dans les sous-balises *<source>* et *<target>*.

La partie *<context>* peut être spécifiée par le concepteur. Ce dernier peut spécifier un contexte dans lequel l'utilisateur est défini comme parlant l'anglais. Ensuite, le concepteur définit les ressources textuelles de chaque *composant abstrait* ou *conteneur abstrait* par rapport au contexte prédéfini. Ces ressources seront utilisées par la suite pour la génération de l'IHM finale.

La Figure IV.36 illustre le code XML complet du modèle AMXML de la sous-tâche *naviguer vers*. Nous pouvons distinguer les parties principales du modèle AMXML, l'intégration des tâches, la description des *conteneurs* et *composants abstraits*, les contextes prédéfinis ainsi que les ressources de l'IHM. Dans la partie suivante, la structure métier du système sera construite à base de composants métier.


```

<amxml>
  <taskModel> // Définir les tâches du système
    <task id="t_naviguer_vers" name="naviguer vers" type="abstract" container="c_naviguer_vers">
      <task id="t_saisir_coordonnees" name="Saisir coordonnées" type="interactive" container="c_saisir_coordonnees">
        <task id="t_choisir_depart" name="Choisir départ" type="interactive" container="c_choisir_depart">
          <task id="t_choisir_type1" name="Choisir type" type="interactive" function="choice" container="com_choisir_type1"/>
          <task id="t_saisir1" name="Saisir" type="interactive" function="input" container="com_saisir1"/>
        </task>
        <task id="t_choisir_arrivee" name="Choisir arrivée" type="interactive" container="c_choisir_arrivee">
          <task id="t_choisir_type2" name="Choisir type" type="interactive" function="choice" container="com_choisir_type2"/>
          <task id="t_saisir2" name="Saisir" type="interactive" function="input" container="com_saisir2"/>
        </task>
      </task>
    <task id="t_afficher_chemin" name="Afficher chemin" type="abstract" container="c_afficher_chemin">
      <task id="t_choisir_planification" name="Choisir planification" type="interactive" function="choice"
        container="com_choisir_planification"/>
      <task id="t_calculer_chemin" name="Calculer chemin" type="automatic" function="calculate" resource="chemin.xml"
        container="com_calculer_chemin"/>
      <task id="t_afficher_carte" name="Afficher carte" type="interactive" function="plan" resource="datapath/carte.data"
        container="com_afficher_carte"/>
    </task>
  </taskModel>
  <relationships> // Définir les liens entre les tâches
    <link id="l_saisir_coordonnees_afficher_chemin" type="enabling_data">
      <source sourceId="t_saisir_coordonnees" out="default"/><target targetId="t_afficher_chemin" in="default"/></link>
    <link id="l_choisir_depart_choisir_arrivee" type="concurrency">
      <source sourceId="t_choisir_depart" out="default"/><target targetId="t_choisir_arrivee" in="default"/></link>
    <link id="l_choisir_type_saisir1" type="enabling">
      <source sourceId="t_choisir_type1" out="default"/><target targetId="t_saisir1" in="default"/></link>
    <link id="l_choisir_type_saisir2" type="enabling">
      <source sourceId="t_choisir_type2" out="default"/><target targetId="t_saisir2" in="default"/></link>
    <link id="l_choisir_planification_calculer_chemin" type="enabling_data">
      <source sourceId="t_choisir_planification" out="default"/><target targetId="t_calculer_chemin" in="default"/></link>
    <link id="l_calculer_chemin_afficher_carte" type="enabling_data">
      <source sourceId="t_calculer_chemin" out="default"/><target targetId="t_afficher_carte" in="default"/></link>
  </relationships>
</taskModel>
  <aiModel> // Définir les conteneurs et les composants abstraits de l'IHM
    <abstractContainer id="c_naviguer_vers" name="naviguer vers">
      <abstractContainer id="c_saisir_coordonnees" name="Saisir coordonnées">
        <abstractContainer id="c_choisir_depart" name="Choisir départ">
          <abstractComponent id="com_choisir_type1" name="Choisir type"/>
          <abstractComponent id="com_saisir1" name="Saisir"/>
        </abstractContainer>
        <abstractContainer id="c_choisir_arrivee" name="Choisir arrivée">
          <abstractComponent id="com_choisir_type2" name="Choisir type"/>
          <abstractComponent id="com_saisir2" name="Saisir"/>
        </abstractContainer>
      </abstractContainer>
    <abstractContainer id="c_afficher_chemin" name="Afficher chemin">
      <abstractComponent id="com_choisir_planification" name="Choisir planification"/>
      <abstractComponent id="com_calculer_chemin" name="Calculer chemin"/>
      <abstractComponent id="com_afficher_carte" name="Afficher carte"/>
    </abstractContainer>
  </aiModel>
  <contextModel> // Définir les contextes prévus
    <context id="co1"><environment id="e1" type="any" workstate="visiting"/><user id="u1" language="EN"/></context>
  </contextModel>
  <resourceModel> // Définir les ressources des composants de l'IHM
    <resource id="r_c_saisir_coordonnees_co1" component="c_saisir_coordonnees" context="co1" text="Input the coordinates"/>
    <resource id="r_com_choisir_type1_co1" component="com_choisir_type1" context="co1" text="Choose a category"/>
    <resource id="r_com_choisir_type1_co1" component="com_choisir_type2" context="co1" text="Choose a category"/>
    <resource id="r_choisir_planification_co1" component="com_choisir_planification" context="co1" text="Choose a planning category"/>
  </resourceModel>
</amxml>

```

Figure IV.36 : AMXML de l'IHM « naviguer vers »

II.4.2 Construction de la structure métier

Dans cette étape, la structure métier est construite (cf. Figure IV.37) grâce aux règles de construction de la structure métier (cf. Chapitre III, §IV.5). Tout d'abord, en appliquant la règle 1, des composants métier "conteneur" sont associés aux *conteneurs abstraits* : *Naviguer vers*, *Saisir coordonnées*, *Afficher chemin*, *Choisir départ*, et *Choisir arrivée*. La règle 3 peut être appliquée sur les *conteneurs abstraits* *Choisir départ* et *Choisir arrivée*, car leurs tâches associées s'exécutent en concurrence. Par conséquent, deux composants métier "panneau" sont associés à ces deux *conteneurs abstraits*. Ensuite les deux composants métier "conteneur" associés à ces deux *conteneurs abstraits* sont supprimés.

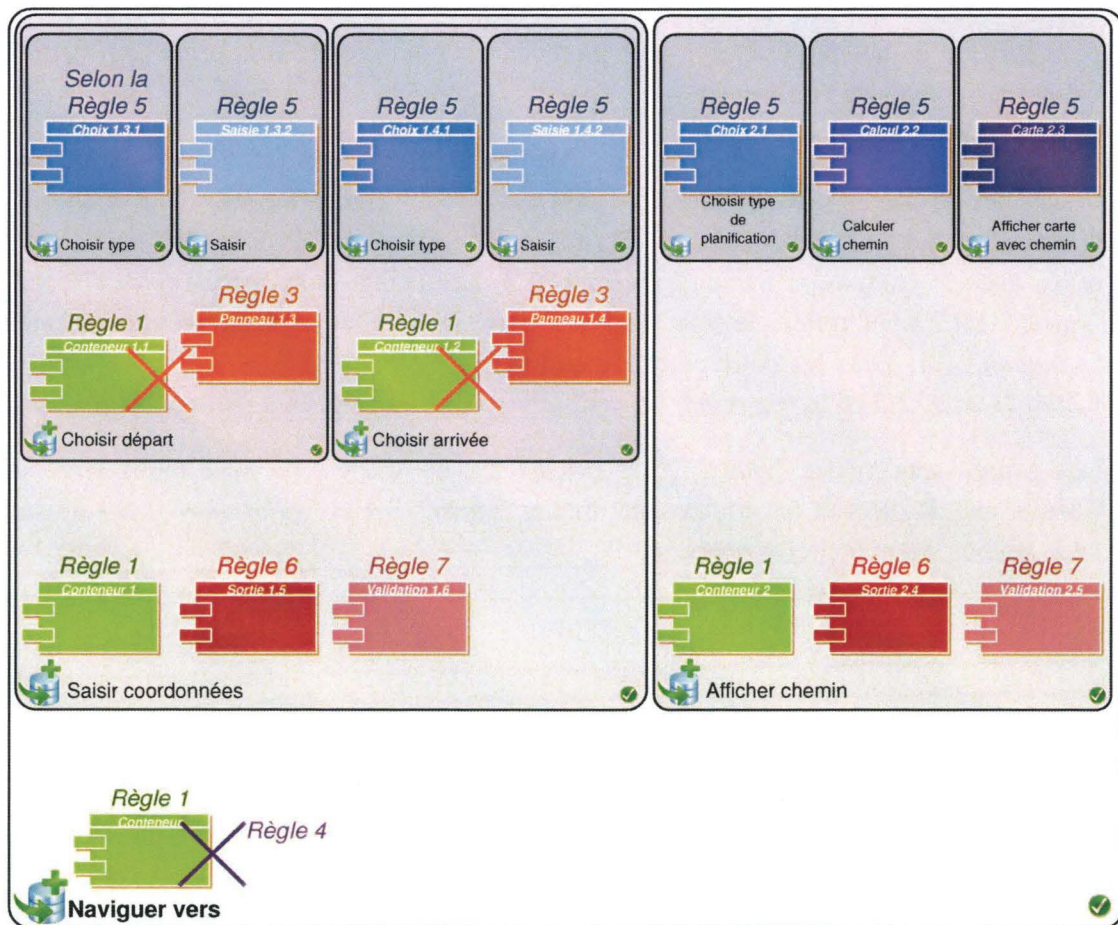


Figure IV.37 : Application des règles de génération de la structure métier

La règle 5 permet d'associer des composants métier adéquats aux *composants abstraits*, à l'aide des patrons de conception. La sélection des composants s'effectue en fonction du type de la tâche associée au *composant abstrait*. Les facettes de présentation des composants métier correspondant sont placées dans la facette de présentation du composant métier associé au *conteneur abstrait* mère. Des composants métier "choix" sont associés aux deux *composants abstraits* *Choiir type*, ainsi qu'au composant *Choiir type de planification*. Deux composants métier "saisie" sont associés aux *composants abstraits* *Saisie*. Un composant métier "calcul" est associé au *composant abstrait* *Calculer chemin*.

Enfin un composant métier "*Carte*" est associé au *composant abstrait Afficher carte avec chemin*.

Comme le *conteneur abstrait naviguer vers* supervise deux *sous-conteneurs abstraits* dont les tâches s'exécutent séquentiellement et des composants métier de type "*conteneur*" sont déjà associés aux sous-conteneurs, le composant métier "*conteneur*" associé à ce conteneur est supprimé, selon la règle 4.

Enfin, des composants métier "*sortie*" (en appliquant la règle 6) et "*validation*" (en appliquant la règle 7) sont associés aux *conteneurs abstraits Saisir coordonnées* et *Afficher chemin* qui ont déjà un composant métier "*conteneur*". Le composant "*sortie*" permettra de quitter le système à tout moment. Le composant "*validation*" permettra de valider les entrées de l'utilisateur.

Les composants métier sélectionnés précédemment, sont reliés (cf. Figure IV.38) en se basant sur la partie *<relationships>* d'AMXML (voir Figure IV.36). Tout d'abord les composants métier "*choix*" 1.3.1 et "*saisie*" 1.3.2 sont reliés. Ensuite tous les deux sont reliés avec le composant métier "*panneau*" 1.3. Les composants métier "*choix*" 1.4.1 et "*saisie*" 1.4.2 sont reliés. Ensuite tous les deux sont reliés avec le composant métier "*panneau*" 1.4. Tous les deux composants métier "*panneau*" 1.3 et 1.4, les composants métier "*sortie*" 1.5 et "*navigation*" 1.6 sont reliés avec le composant métier "*conteneur*" 1.

Les composants métier "*choix*" 2.1, "*calcul*" 2.2 et "*carte*" 3.3 sont reliés entre eux. Ensuite ces derniers et les composants métier "*sortie*" 2.4 et "*validation*" 2.5 sont reliés avec le composant métier "*conteneur*" 2.

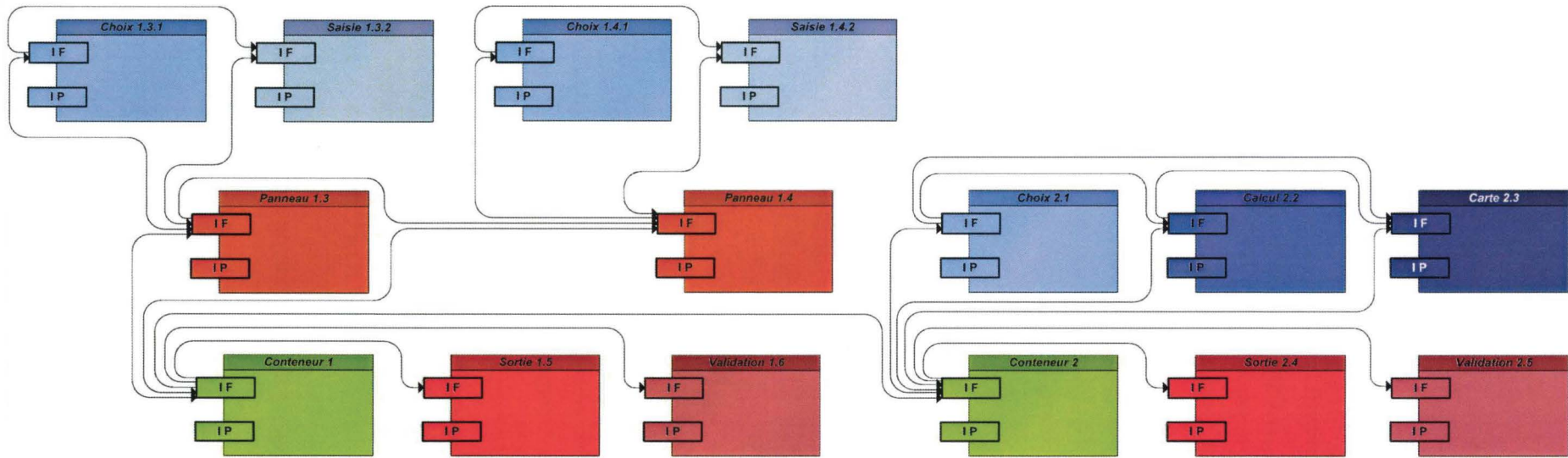


Figure IV.38 : Structure métier « Naviguer vers »

II.4.3 Adaptation primaire de l'IHM

Pendant l'adaptation primaire, les composants de présentation sont sélectionnés (cf. Figure IV.39) selon le contexte d'usage prédéfini par le concepteur (utilisateur anglais, plate-forme de type PDA, modalité graphique).

Tout d'abord, les composants métier "*choix*" 1.3.1, 1.4.1 et 2.1 choisissent des composants de présentation de type *RadioButton*. L'espace occupé par ce type de composants de présentation convient avec la plate-forme de type PDA (taille d'écran moyenne, et écran tactile pour l'interaction) ; de plus, nous avons moins de cinq choix.

Ensuite, les deux composants métier "*saisie*" 1.3.2 et 1.4.2 choisissent des composants de présentation de type *EditBox* pour la saisie. Puis, les composants "*sortie*" et "*validation*" choisissent deux composants de type *bouton*, un pour quitter le système et l'autre pour valider le choix et/ou la saisie de l'utilisateur.

Un composant de présentation multicouche, dédié à l'affichage de la carte, peut être utilisé. La notion de conception en multicouches (Gustavsson-Christiernin et Torgersson, 2005) peut être intégrée dans l'architecture du composant de présentation. Ce dernier peut posséder plusieurs couches de présentation. Chaque couche est responsable de l'affichage de telle information (par exemple, les stations de bus, les gares, etc.). Ensuite les couches peuvent être affichées ou cachées en fonction du contexte d'usage (par exemple l'âge de l'utilisateur). Dans le cas d'un enfant, l'interface doit être simple et très facile à comprendre, avec des dessins animés (si la plateforme cible supporte ce type d'affichage) ; l'IHM montre clairement les endroits les plus intéressants selon le point de vue des enfants tels que les parcs avec jeux. Elle ne contient pas d'informations détaillées ou de mots trop spécifiques. Enfin les couleurs de l'IHM doivent être adaptées aux enfants (il est important de se baser à ce sujet sur les recommandations des ergonomes²²). Dans le cas d'un adulte, l'IHM propose plus de détails en ce qui concerne le chemin à suivre, la localisation des parcs d'attraction, des banques, de la poste, des offices de tourisme, etc. (cf. Figure IV.40).

Des composants de présentation de type fenêtre sont choisis pour les composants métier "*conteneur*". La fenêtre spécifique à la tâche *Saisir coordonnées* est affichée d'abord à l'utilisateur. Elle contient deux composants de présentation *Button* associés aux composants métier 1.5 et 1.6. Elle contient également deux composants de présentation de type *panneau* dédiés aux deux composants métier "*panneau*" 1.3 et 1.4. Ces derniers contiennent les composants de présentation déjà sélectionnés pour les composants métier 1.3.1, 1.3.2, 1.4.1 et 1.4.2.

²² Notons aussi le courant de recherche "CHI Kids" du Special Interest Group de l'ACM (SIGCHI) [www.sigchi.org].

Après les saisies de l'utilisateur dans la fenêtre *Saisie coordonnées*, lors de la validation, cette fenêtre est cachée et une deuxième fenêtre *Afficher chemin* associée au composant métier 2 est affichée. Cette fenêtre présente progressivement les facettes de présentation des composants métier associés à son composant métier. La fenêtre contient d'abord les composants de présentation *RadioButton* associés au composant métier 2.1. Ensuite grâce au composant de présentation *Button* associé au composant métier "validation" 2.3, les composants *RadioButton* sont cachés, et la facette de présentation du composant métier "calcul" 2.2 est affichée. Cette dernière se présente sous la forme d'un message demandant à l'utilisateur de patienter pendant le calcul du chemin. Le bouton du composant métier "validation" est caché puisqu'il n'y a pas de saisie ou de choix à effectuer par l'utilisateur.

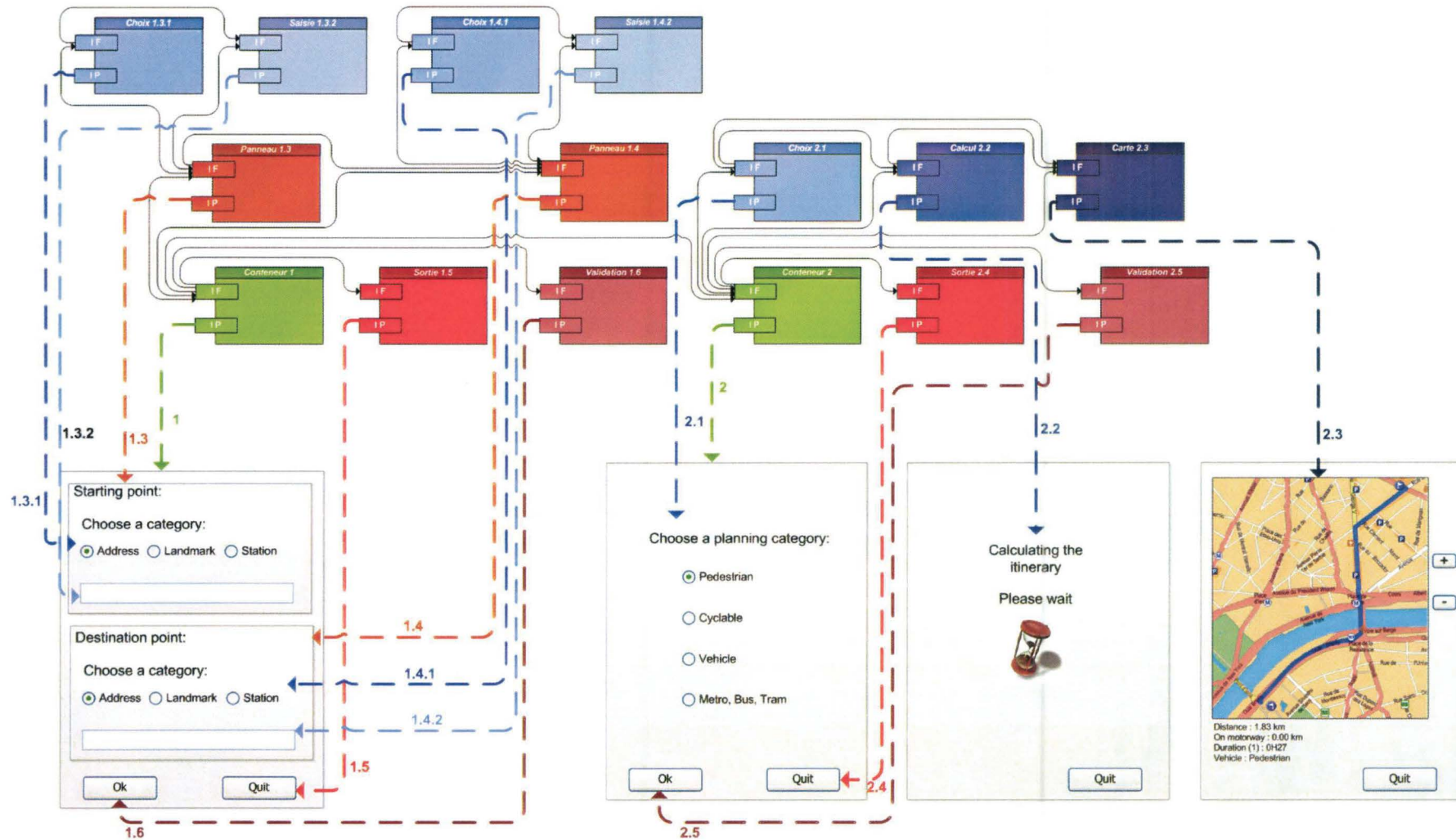


Figure IV.39 : première version du système de guidage touristique « Naviguer vers »

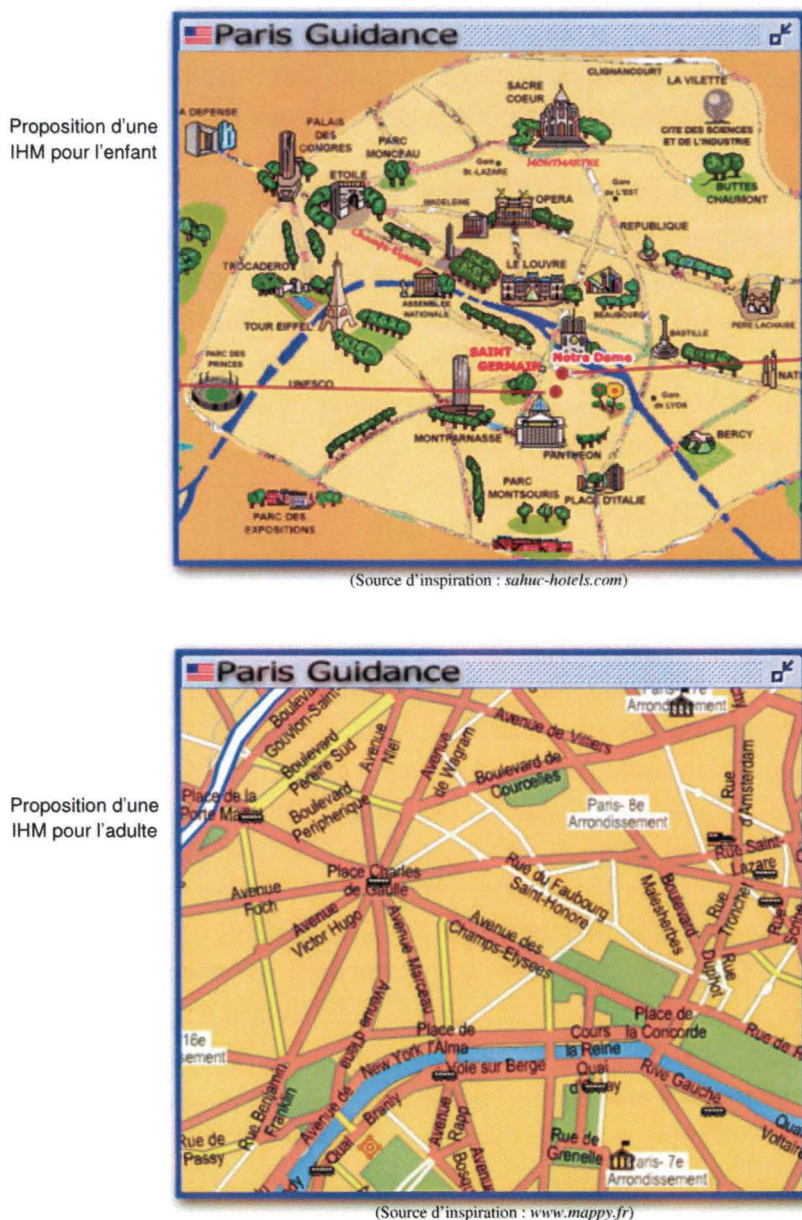


Figure IV.40 : Partie de l'IHM du système de guidage touristique adaptée à la catégorie d'âge du visiteur

En effet, la facette de présentation du composant métier "validation" n'est affichée qu'en cas de saisie ou de choix présenté dans la facette de présentation du composant métier "conteneur" associé au conteneur abstrait mère.

Enfin le message d'attente (concernant la facette de présentation de composant métier "calcul" 2.2) est caché, et le composant de présentation sélectionné au composant Afficher carte est affiché. L'utilisateur peut quitter à tout moment la tâche Naviguer vers en appuyant sur le bouton Quit (sélectionné pour le composant métier "sortie" 2.5).

Avant la distribution du système sur le dispositif cible, le système est équipé des processus d'apprentissage et de capture du contexte. A la fin de cette étape, l'IHM finale sera prête à être distribuée sur la plate-forme cible et à être exploitée par l'utilisateur final.

II.4.4 Utilisation du système interactif plastique

Le système interactif commence à être exploité sur la plate-forme cible par l'utilisateur final. Les tâches s'effectuent par l'intermédiaire des composants fonctionnels (ils sont encapsulés à l'intérieur des composants métier, cf. chapitre III, §IV.1). Le contexte d'usage est actualisé avec des informations capturées concernant l'utilisateur, la plate-forme et l'environnement. L'étape d'adaptation vivante peut être appelée lors d'un éventuel changement contextuel.

II.4.5 Adaptation vivante

Un exemple de changement du contexte concerne le mode d'affichage sur le PDA, ce dernier possède un équipement qui permet de capturer certains types de mouvements de l'appareil. Par conséquent, l'affichage peut changer entre *vertical* et *horizontal* en fonction de la situation de l'appareil dans les mains de l'utilisateur. L'IHM est précédemment générée et adaptée à un affichage vertical. Donc lors du passage au mode horizontal, tous les composants de présentation sont éventuellement réarrangés. De plus, le patron de conception "arrangeur" qui gère le positionnement de chaque composant, peut, par exemple, changer la stratégie d'arrangement du groupe des composants de type *RadioButton* pour qu'il soit effectué verticalement comme ceux situés dans la fenêtre *Saisir coordonnées*, ou horizontalement et verticalement, comme le cas des composants situant dans la fenêtre *Choisir type de planification* (cf. Figure IV.41). Nous constatons aussi que lors du passage d'un modèle d'affichage vertical à celui horizontal, les panneaux dans la fenêtre *Saisir coordonnées* sont réarrangés horizontalement en répondant au changement du mode d'affichage.

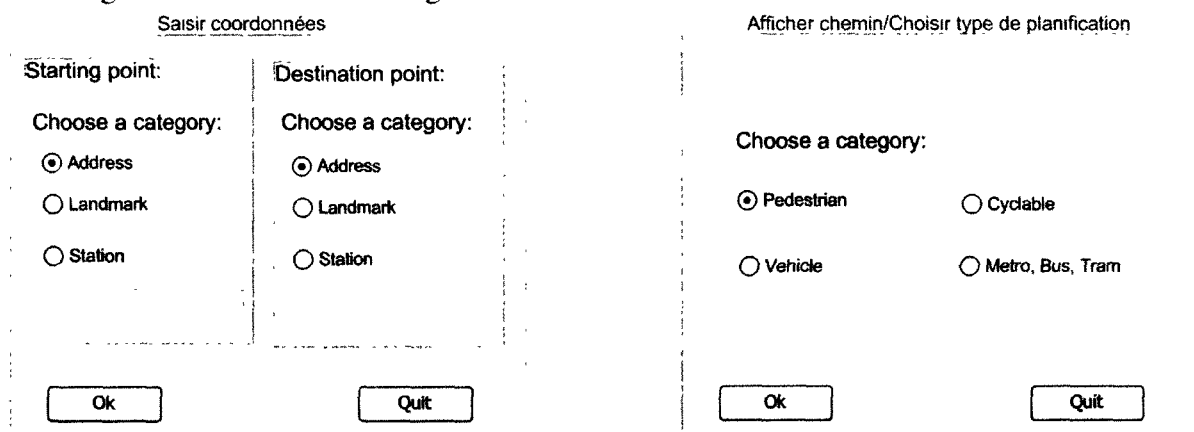


Figure IV.41 : Changement de l'orientation d'affichage de PDA

II.4.6 Processus d'apprentissage

Les processus d'apprentissage travaillent avec le système afin d'améliorer la qualité des décisions à prendre à l'exécution et parfois de celles relatives à la conception. L'IHM est

équipée de mécanismes donnant la possibilité à l'utilisateur de déplacer les facettes de présentation, de changer les couleurs de l'IHM, changer la police de texte, ou choisir d'autres types de composants de présentation pour les facettes de présentation. Par exemple, lors de l'utilisation du système par un touriste venant du Moyen-Orient, le touriste clique sur le bouton *Sortie* à la place de *Validation*, car il a l'habitude que le bouton *validation* s'affiche à droite ; ensuite l'utilisateur relance le système et relocalise les deux boutons (cf. Figure IV.42). Cette réaction de l'utilisateur est enregistrée et ajoutée à la base de connaissance ; autrement dit le patron responsable de l'arrangement des composants de présentation, changera sa stratégie en prenant en compte l'origine du touriste.

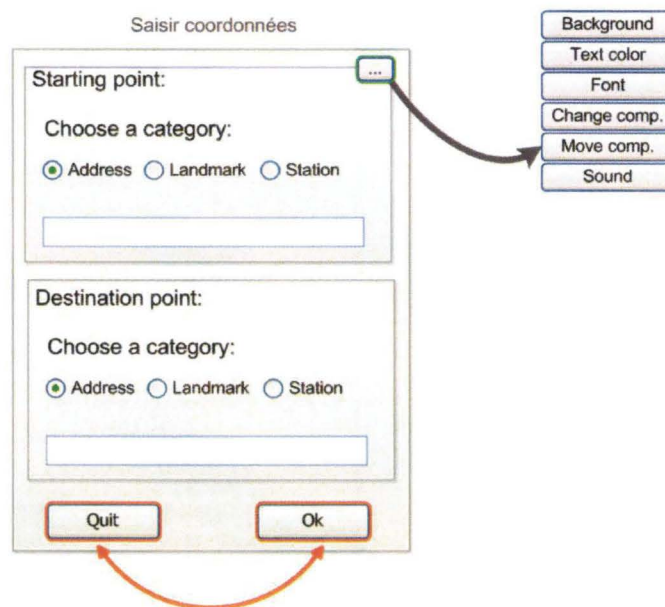


Figure IV.42 : changement de localisation des boutons *Quit* et *Ok*

Les Figure IV.44 et Figure IV.44 illustrent l'arbre de décision (généré avec l'algorithme C4.5) de la partie du patron *Arrangeur* concernant la localisation des composants de présentations *Validation* et *Sortie*. Au départ la localisation de ceux-ci était relative au métier du composant. Après l'apprentissage, l'origine a été prise en compte pour ces deux composants et a été intégré dans la partie décisionnelle du patron *Arrangeur*.

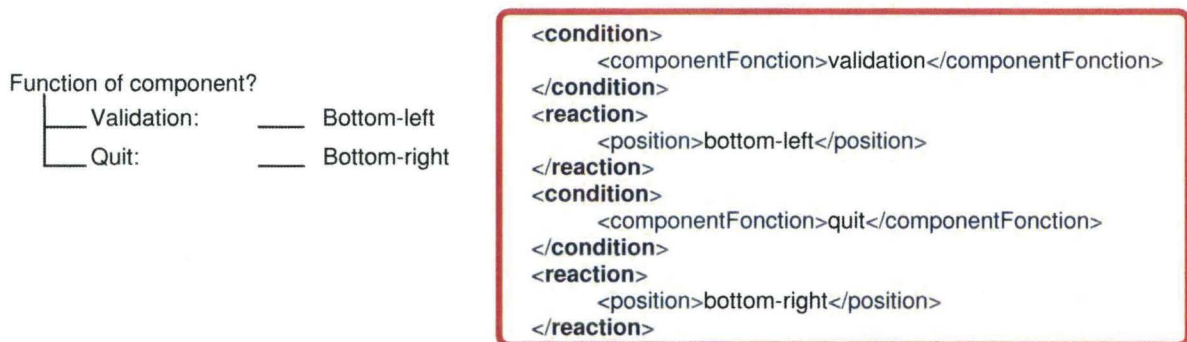


Figure IV.82 : Une partie de l'arbre de décision du patron *arrangeur*, avant le processus d'apprentissage



Figure IV.44 : Une partie de l'arbre de décision du patron arrangeur, après le processus d'apprentissage

II.5 Conclusion sur le système de guidage touristique

Dans un but d'étude de faisabilité, la méthode a été appliquée sur un premier cas d'étude dont le contexte d'usage est varié et fréquemment changé. Le modèle AMXML a été construit à partir du modèle de tâche grâce aux règles de construction d'AMXL proposées. Ensuite le système de guidage touristique a été généré à partir de ce modèle. La représentation des *conteneurs et composants abstraits* de l'IHM en format graphique d'UsiXML peut faciliter la structure abstraite de l'IHM.

Une version concrète à base de composants métier a été générée en appliquant les règles de construction de la structure métier. La première version du système de guidage touristique a été générée pour un contexte d'usage prévu par le concepteur. Ensuite à l'exécution, nous avons rencontré la nécessité de l'adaptation vivante lors de changement du mode d'affichage. Une mise à jour de la base de connaissance a été effectuée suite à une réaction d'un utilisateur. Nous allons présenter par la suite un second cas d'étude en lien avec un système industriel de fabrication de godets métalliques. Nous allons appliquer à nouveau notre méthode dans ce cadre.

III. Second cas d'étude : Système de supervision d'une usine chimique

Dans cette seconde étude de cas, nous appliquons notre méthode à un système de supervision industrielle. Tout d'abord, nous abordons le scénario du système, avec les différentes facettes du contexte d'usage envisagé. Ensuite, des spécifications du système en TOOD [Tabary, 2001 ; Mahfoudhi *et al.*, 2005] sont converties afin de construire notre propre modèle AMXML. En effet, TOOD est une méthode de spécification et conception de système interactif, basée sur la tâche ; elle nous permet d'avoir des spécifications rigoureuses sur les tâches du système et de l'utilisateur dans le domaine industriel et particulièrement celui de supervision et de contrôle.

Ensuite, des composants métier sont sélectionnés en appliquant des règles de construction de la structure métier, et sont reliés. L'IHM est générée sous la forme d'une première version à base de composants de présentation sélectionnés selon le contexte prédéfini par le concepteur. A l'exécution, nous obtenons des résultats de l'intégration de la notion d'apprentissage et d'adaptation vivante dans la méthode.

III.1 Présentation de l'application industrielle

L'application concerne le cas d'un procédé industriel de fabrication de godets métalliques remplis par une solution chimique préparée préalablement. Le scénario du système est adapté de [Riahi, 2004 ; Moussa *et al.*, 2006]. Le processus simplifié de préparation comprend deux phases principales (cf. Figure IV.84) :

- une première phase qui consiste à façonner les godets par emboutissage de pièces métalliques prédécoupées ;
- une deuxième phase qui consiste à verser dans les godets une dose de solution chimique. La solution est obtenue à partir d'une préparation primaire P et d'un produit S.

L'étude a porté sur cette dernière phase qui met en œuvre deux modules : un module de préparation et un module de remplissage.

Le module de remplissage comprend principalement un réservoir de solution et un doseur de remplissage. Les godets façonnés sont présentés un à un pour remplissage. L'arrivée d'un godet à l'emplacement d'un remplissage est détectée par une cellule photoélectrique. La commande du doseur de remplissage est alors activée jusqu'à la réception d'un signal impulsif de fin de remplissage. Une fois rempli, le godet est évacué automatiquement et un nouveau godet à remplir prend sa place. La présentation des godets vides et leur évacuation une fois remplis sont supposées être commandées par un module à part.

Le réservoir de solution est équipé d'une électrovanne de sécurité pour la commande de vidange du réservoir et de trois détecteurs de niveau :

- un détecteur de niveau minimum : n_r ($n_r = 1$ ssi le niveau minimum de la solution est atteint sinon $n_r = 0$),

- un détecteur de niveau maximum : x_r ($x_r = 1$ ssi le niveau maximum de la solution est atteint sinon $n_r = 0$),
- un détecteur de niveau seuil réapprovisionnement : r_r ($r_r = 1$ ssi le niveau seuil de réapprovisionnement existe).

Le module de préparation est équipé principalement de deux doseurs similaires à celui de remplissage pour le dosage des produits P et S, d'une cuve, d'un moteur pour agiter la solution et d'une électrovanne pour commander le remplissage de la solution dans le réservoir (Figure IV.84).

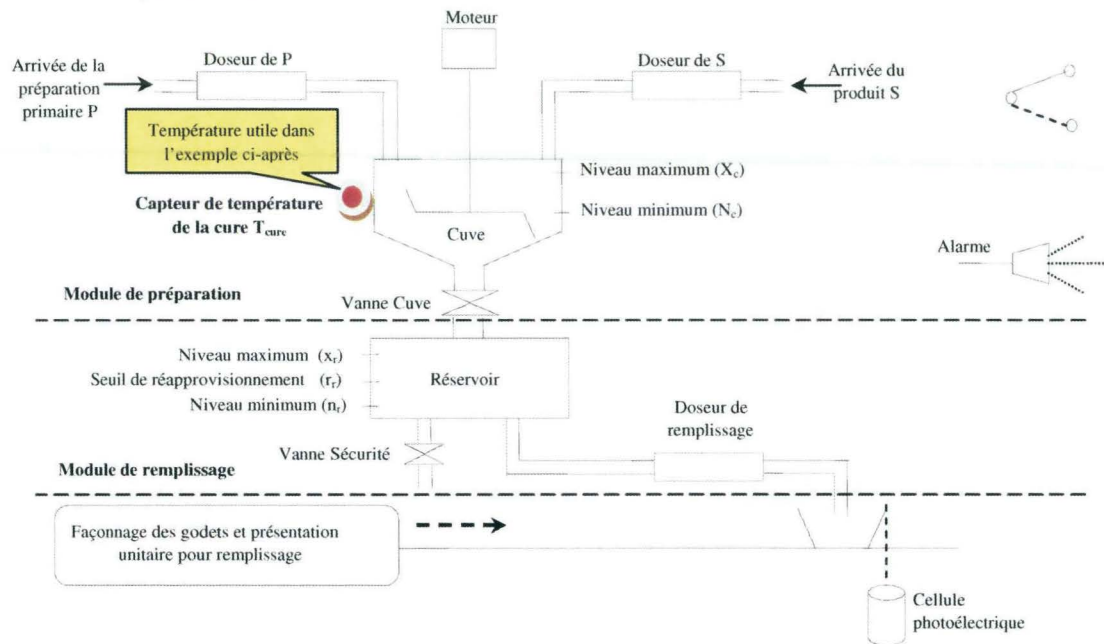


Figure IV.84 : Façonnage et remplissage de godets

A chacun des deux doseurs de ce module sont associés deux signaux de commande pour indiquer la Début et la Fin de dosage (DP et FP pour le dosage du produit P ; DS et FS pour le dosage du produit S). Aux deux doseurs sont également associés deux dispositifs permettant de régler les quantités de dosage respectives et des dispositifs de chauffage et refroidissement permettant d'ajuster les températures des solutions. La cuve est équipée de deux détecteurs de niveau :

- un détecteur de niveau minimum : N_c ($N_c = 1$ ssi le niveau minimum est atteint),
- un détecteur de niveau maximum : X_c ($X_c = 1$ ssi le niveau maximum est atteint),

Trois thermomètres sont prévus pour mesurer les températures de la solution primaire P, du produit S et de la solution de mélange.

Le module de préparation est commandé par un relais à deux positions (marche/arrêt). Dès la mise du relais en position marche, le module de préparation se met à développer deux fonctions parallèles :

- d'un côté, le moteur est commandé pour agiter la solution ;

- d'un autre côté, chaque fois que le niveau de réapprovisionnement atteint ou passe en dessous du réservoir disparaît ($r_r = 0$), il y a déclenchement d'un cycle de préparation-réapprovisionnement du réservoir. A savoir, la préparation de la solution chimique se déroule en injectant dans la cuve d'une dose du produit P, puis une dose du produit S est injecté, enfin la commande de l'électrovanne est envoyée pour le remplissage du réservoir jusqu'au niveau maximum.

L'opération de remplissage n'est autorisée que si la température de la cure T_{cure} ne dépasse pas les 55°C. Cette température est relative aux températures des composants P et S utilisés pour la préparation du mélange chimique.

Il est question de supervision, de détection d'alerte, de commande et de suivi. Nous supposons que l'ensemble de ces tâches est attribué à un ou plusieurs opérateurs humains.

III.2 Contextes d'usage envisagés

Nous aborderons, dans cette partie, les différentes facettes du contexte d'usage à envisager. L'IHM plastique du système de supervision doit les prendre en compte et observer les changements contextuels à l'exécution.

Nous verrons que les tâches de l'utilisateur (contexte relatif à l'utilisateur) peuvent être indiquées ou modifiées en fonction des données de contrôle, autrement dit les quantités mesurées (contexte environnemental).

III.2.1 Contexte relatif à l'utilisateur

L'utilisateur du système interactif visé (cf. Tableau IV.6) peut être un opérateur dans la salle de contrôle qui peut intervenir directement selon les autorisations données :

- le superviseur en chef qui administre et visualise tous les synoptiques de toutes les sections, et en informe les opérateurs responsables des processus en cas de nécessité ;
- les experts d'astreinte qui peuvent surveiller et intervenir à distance (éventuellement de chez eux) ;
- et les opérateurs avec un profil particulier qui leur permet d'accéder à certaines informations et peuvent intervenir de manière très limitée.

L'IHM doit prendre en compte la catégorie de l'utilisateur, et respecter les autorisations données à chacune et celles temporaires qui peuvent être données en cas d'urgence.

Certains utilisateurs travaillent en mobilité (opérateurs nomades), tandis que d'autres partagent leur temps entre la salle de contrôle et le terrain. Ils peuvent être en mobilité intra-entreprise (au sein d'un site de l'entreprise) ou extra-entreprise, ils travaillent alors à distance et accèdent au système de l'extérieur en cas de nécessité.

Chaque utilisateur peut spécifier ses préférences concernant l'IHM du système (couleurs préférées, mode d'interaction favori,...). De plus, pour un utilisateur possédant un

éventuel handicap, l'IHM doit pouvoir offrir des couleurs spéciales pour le fond d'écran et pour les composants de présentation, des messages sonores et/ou des alertes vibrantes selon le type de handicap.

Tableau IV.6 : Facettes du contexte relatif à l'utilisateur

Type d'opérateur	Autorisations	Mobilité	Heures de travail	Mode d'interaction préféré	Couleurs préférées	handicap
Superviseur en chef	Arrêt des machines	Non		Vocal	Noir/blanc	Non
Superviseur (Opérateur en salle de contrôle)	Réglage des machines	Nomade	Créneau horaire (ex : 8h-15h)	Tactile	Bleu/blanc	Visuel
Opérateur d'astreinte	Intervention à distance	Intra-entreprise		Clavier	...	Auditif
Opérateur avec profil particulier		extra-entreprise		Joystick		...
				...		

III.2.2 Contexte relatif à l'environnement

Les caractéristiques de l'environnement sont variées et peuvent changer pour certains d'entre elles fréquemment (cf. Tableau IV.7). L'IHM doit prendre en compte une gamme large de facettes liées au contexte relatif à l'environnement. Le système interactif peut être utilisé dans la salle de contrôle, à l'intérieur de l'entreprise, ou dans la voiture ou au domicile de l'opérateur d'astreinte qui peut se connecter au système et intervenir à distance. La stabilité du niveau de luminosité est variée selon le lieu de travail.

L'environnement de travail peut être très bruyant (à côté des machines, ou dans la voiture), un peu bruyant (dans la salle de contrôle si plusieurs opérateurs y travaillent dans des situations où les communications sont alors nécessaires), ou calme (bureau du superviseur en chef). Dans un environnement bruyant, le système interactif peut être contraint d'utiliser (ou proposer) un niveau sonore élevé avec des alertes vibrantes. En revanche, en environnement calme, il s'agit de baisser le son, utiliser des alertes vibrantes et/ou des témoins lumineux.

Les opérateurs nomades sont équipés de dispositifs mobiles disposant de fonctions de communication et de localisation. Cela permet de localiser les opérateurs à tout moment ; par exemple : le système de supervision peut localiser les opérateurs mobiles dans un voisinage ou zone donné, et leur envoyer des alertes basées sur la proximité de certains événements (avec localisation et type).

Les données de contrôle (comme la température, l'alimentation électrique, l'état des machines,...) peuvent jouer un rôle très important à changer les tâches ou les autorisations de l'utilisateur. Ces données doivent être représentées selon un ou plusieurs types d'échelles de mesure, l'utilisateur ayant à prendre des décisions s'appuyant sur ces données de contrôle.

Tableau IV.7 : Facettes du contexte relatif à l'environnement

Lieu	Luminosité	Bruit	Heure	Etat	Situation	Données de contrôle (ex. température)
Salle de contrôle	Basse	Bruyant		Urgent	Travail	35°
Section I	Moyenne	Calme	Valeur au format heure (ex : 12 :00)	Normal	Hors-travail	...
Extérieur	élevée			...	occupé	...

Nous allons détailler les échelles de mesure utilisées par l'IHM afin de présenter les données de contrôle selon différents modes d'affichage.

Le principe des échelles de mesure a été introduit par [Stevens, 1946] qui a distingué quatre types d'échelles de mesure : nominal, ordinal, intervalle et de rapport. Ces échelles sont utilisées pour la représentation de quantités mesurées. La distinction entre ces types de représentation des données repose sur la quantité ou sur les caractères qualitatifs de l'information exprimée par les données. Les différents types de données sont décrits au Tableau IV.8.

Tableau IV.8 : Échelles de mesure [Stevens, 1946]

Échelles de mesure	Description	Exemples
ECHELLES DE RAPPORT (valeurs totales)	Les échelles permettent non seulement la comparaison d'intervalles, mais également la comparaison de rapports (il est possible de déterminer si deux rapports sont ou ne sont pas égaux). Le zéro a une signification précise, puisqu'il désigne l'absence du caractère considéré.	1, 2, 3...
INTERVALLES (valeurs relatives)	Les échelles intervalles permettent de comparer des intervalles (il est possible de déterminer si deux intervalles sont ou ne sont pas de même étendue). Le zéro est situé de manière arbitraire.	Double, trois fois,...
ORDINALES (classement par rang)	Les échelles ordinales identifient les importances relatives, mais ne quantifient pas les différences entre des valeurs. Les échelles ordinales comprennent des données numériques groupées en classes.	Faible < Moyen < Elevé Village < Ville < Cité
NOMINALES (noms, caractère unique)	Les échelles nominales distinguent un article d'un autre, mais ne classent ni ne quantifient les données. Les valeurs nominales devraient s'exclure mutuellement et peuvent être exhaustives.	Bleu, blanc, rouge

Les différents types d'échelles de mesure forment un ordre partiel basé sur les ensembles de transformations permises (cf. Figure IV.85). Les données nominales expriment moins d'information, habituellement rien de plus qu'une distinction entre une donnée et une

autre, tandis que les données de rapport expriment plus d'information, incluant la valeur d'une variable quelconque mesurée sur une échelle absolue.

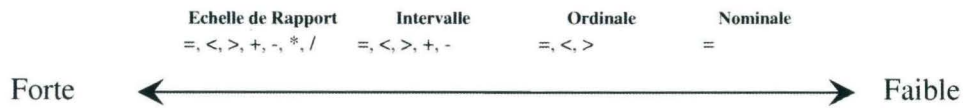


Figure IV.85 : Ordre partiel de types d'échelles [Petersen et May, 2003]

Il est possible d'effectuer des transformations des données représentées en échelle de mesure plus forte vers celle plus faible [Petersen et May, 2006]. Cependant, une fois la transformation d'échelle de mesure effectuée, il peut être possible de faire une transformation vers un niveau plus haut qui pourrait restaurer l'initiale forte échelle de mesure. En cas d'impossibilité de faire la transformation vers un niveau plus haut, les données originales stockées dans la base de données provisoires du système peuvent être récupérées. Le Tableau IV.9 illustre les valeurs possibles de la température de machine T_{cure} (voir Figure IV.84) sur différentes échelles de mesure avec les transformations entre les échelles. Les transformations peuvent être effectuées afin de représenter les données en différents types d'échelle de mesure. Les données initiales de la température sont d'abord représentées en échelle de rapport.

Lorsque la valeur de la température T_{cure} est reçue par le système, en échelle de rapport, toutes les transformations décrites ci-dessous peuvent être légitimement appliquées aux données. Par exemple, les données en échelle ordinale indiquent une classification des données en respectant les catégories ordonnables discrètes (basse, normale, élevée). Nous pouvons décider dans quelle catégorie se trouve la température T_{cure} est dans quelle catégorie, en transformant les données initiales de la température en une échelle de mesure ordinale. De plus, s'appuyant sur les données ordinales, il est possible de juger si deux indications sont égales (par exemple, tous les deux sont normales) et qui est la plus (moins) grande que l'autre.

Tableau IV.9 : Valeurs éventuelles de la température T_{cure} selon différentes échelles de mesure (adapté de [Petersen et May, 2006])

Echelle de mesure	Valeurs possibles	Transformation
Echelle de rapport	X [0,1,2, ... ,100°]	Non
Intervalle	X' [-50, ..., -1,0,1, ... ,50°]	$X' = X + a$
Ordinale	X'' [bas, normal, élevé]	If $X \leq 15$ then $X'' = \text{bas}$ If $25 < X < 55$ then $X'' = \text{normal}$ If $X > 55$ then $X'' = \text{élevé}$
Nominale	X''' [ok, ~ok]	If $25 < X < 55$ then $X''' = \text{ok}$ else $X''' = \sim\text{ok}$

III.2.4 Contexte relatif à la plate-forme

Les opérateurs nomades, de même que ceux procédant à la surveillance des installations de l'entreprise (sans être localisés en salle de contrôle), peuvent être munis d'ordinateurs ultra-portables, de PDA, ou de téléphones portables. Les autres dans la salle de contrôle

peuvent travailler sur des plateformes associées à plusieurs écrans d'affichage. Chaque plate-forme a ses propres caractéristiques. Le processus d'adaptation doit prendre en compte les différentes caractéristiques de la plate-forme cible (cf. Tableau IV.10). Dans ce type de système, nous pouvons envisager la mise en œuvre de la notion de migration de l'IHM ; par exemple, lorsque l'opérateur arrive en salle de contrôle, il continue son travail sur des machines dans la salle, ou l'inverse lorsqu'il quitte la salle, il doit récupérer des informations ou données sur son dispositif portable. Dans ce cas, l'adaptation vivante doit permettre une adaptation à la nouvelle plate-forme.

Tableau IV.10 : Facettes du contexte relatif à la plate-forme

Type	Système d'exploitation	Connexion réseau	Système de localisation	Mode d'affichage	Espace d'affichage	Couleurs	Modes d'interaction disponibles	Style
PC	Windows	Non	Non	Horizontal	220*300	4096	Clavier	Fenêtre, Menu, ...
PDA	Symbian	Wifi	Ad-hoc	Vertical	640*480	65536	Clavier/Joystick	Page Web, bouton, ...
Téléphone cellulaire	Linux	Ad-hoc	GPS		Ecran tactile	Vocal
...

III.3 Application de la démarche

Dans cette partie, nous appliquons notre démarche sur le système de supervision industrielle. Nous supposons cette fois que nous partons d'un modèle de tâche spécifié en TOOD [Tabary, 2001 ; Mahfoudhi *et al.*, 2005], les règles de construction d'AMXML seront appliquées au fur et à mesure sur ce modèle afin de générer le modèle AMXML. Pendant la construction, le concepteur peut intervenir pour spécifier, par exemple, le contexte d'usage prédéfini et les ressources de l'IHM. La structure métier sera ensuite générée en appliquant une autre catégorie des règles. Puis à l'aide de l'expérience des patrons de conception, les composants métier sont reliés. Une version primaire du système sera générée, prenant en compte le contexte prévu, et sera prête à être distribuée vers le dispositif cible. L'adaptation vivante sera appelée à l'exécution lors d'un changement contextuel. L'adaptation vivante permettra de changer la présentation des quantités mesurées (température, vitesse, etc.) sans nécessité de revenir à la conception suite à un important changement dans les quantités mesurées, ou un éventuel changement du type de tâche de l'utilisateur (contexte relatif à l'utilisateur). Enfin, les tâches de l'utilisateur peuvent être modifiées à l'exécution en fonction des valeurs des quantités mesurées (par exemple, la température T_{cure} atteint la valeur critique).

III.3.1 Construction d'AMXML

Comme le système de supervision est vaste, nous nous intéresserons seulement à la tâche *Recevoir alerte* du système de supervision. Cette partie du système est déjà spécifiée dans la méthode TOOD. La Figure IV.86 indique que la tâche principale T1 *Recevoir alerte* se

compose principalement de deux sous-tâches : T1.1 *Afficher alerte* assure la présentation de la donnée de contrôle concernant l'alerte reçue et un éventuel message, et la deuxième T1.2 *Régler paramètres* permet de mettre au point des paramètres qui pourraient permettre de résoudre certains problèmes à l'origine de l'alerte reçue. Le modèle ci-dessous définit l'utilisation de l'ensemble des types de données et décrit les traitements appliqués à ces données. L'objet TOOD T1 *Recevoir alerte* possède deux interfaces d'entrée/sortie :

- Interface d'entrée (à gauche) : deux entrées (paramètre à régler, et valeur pour le paramètre choisi), un déclenchement (recevoir alerte) et cinq contrôles (nom et valeur de donnée de contrôle, message, liste de paramètres modifiables, et liste de réactions autorisées)
- Interface de sortie (à droite) : une sortie (Liste des paramètres modifiés).

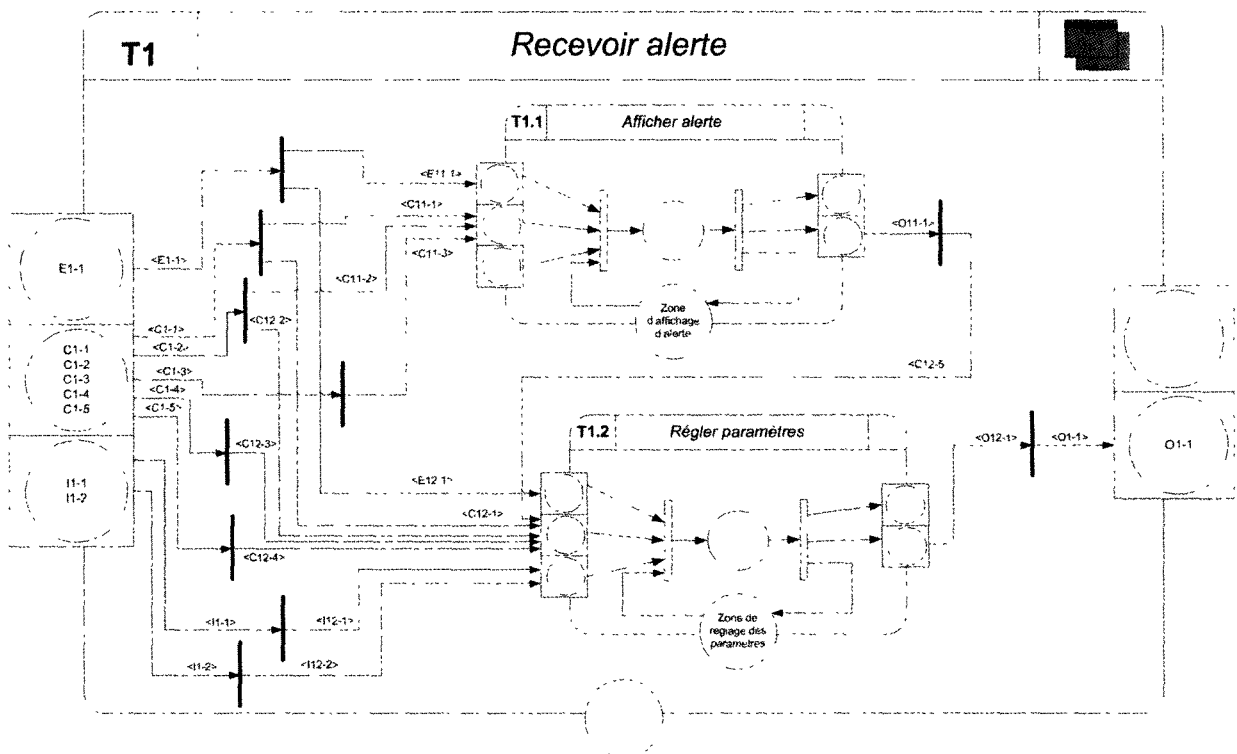


Figure IV.86 : Modèle de tâche dynamique de la tâche « Recevoir alerte » en TOOD²³

La Figure IV.87 illustre une description de l'interface d'entrée/sortie de la tâche *Recevoir alerte*. La tâche T1.1 peut être lancée par l'événement E11-1 (événement d'origine E1-1) correspondant à une alerte reçue. Les contrôles C11-1 et C11-2 concernant les données de contrôle ainsi que C11-3 concernant le message d'alerte reçu, seront prises en compte pour effectuer la tâche *Afficher alerte*. La donnée de sortie O11-1 informe la sous-tâche *Régler paramètres* que l'alerte est bien présentée à l'utilisateur.

²³ La compréhension dans ses moindres détails de la Figure IV.86 n'est pas nécessaire dans le cadre de cette étude de cas. Le lecteur intéressé par une description complète de la méthode TOOD pourra se référer à [Tabary, 2001]

La tâche T1.2 peut être lancée par l'événement E12-1 (événement d'origine E1-1) correspondant à une alerte reçue (même événement que pour la précédente sous-tâche). L'interface d'entrée/sortie de cette tâche est connectée aux contrôles :

- C12-1 et C12-2 concernant le type et la valeur des données de contrôle,
- C12-3 est relatif à la liste des paramètres autorisés à être modifiés par l'opérateur humain actuel,
- C12-4 concernant la liste des réactions autorisées (valeurs possibles pour le paramètre choisi par l'opérateur),
- et C12-5 indique que l'alerte est présentée à l'opérateur.

Enfin la donnée de sortie O12-1 contient la liste des paramètres réglés.

Nom : T1 recevoir alerte	
Description : recevoir des alertes, les afficher, et permettre à l'opérateur humain concerné de prendre des décisions.	
Classe composite : Système de supervision industrielle	
Classes composantes :	T1.1 Afficher alerte T1.2 Régler paramètres
Evénements	
	E1-1 : Recevoir alerte
Contrôle	
	C1-1 : Nom de donnée de contrôle
	C1-2 : Valeur de donnée de contrôle
	C1-3 : Message
	C1-4 : Liste de paramètres modifiables
	C1-5 : Liste de réactions autorisées
Entrées	
	I1-1 : Paramètre à régler
	I1-1 : Valeur pour le paramètre choisi
Sortie	
	O1-1 : Liste des paramètres modifiés
Réaction	
	Néant

Figure IV.87 : Fiche descriptive de la tâche « Recevoir alerte »

La Figure IV.88 illustre les deux zones d'interaction correspondant aux deux tâches *Afficher alerte* et *Régler paramètres*. Ces zones décrivent les activités à travers l'interface à un haut niveau d'abstraction. Elles décrivent également les états, les actions et les déclenchements nécessaires à l'exécution des tâches. La description des activités dans les zones d'interactions s'effectue en utilisant les réseaux de pétri objet.

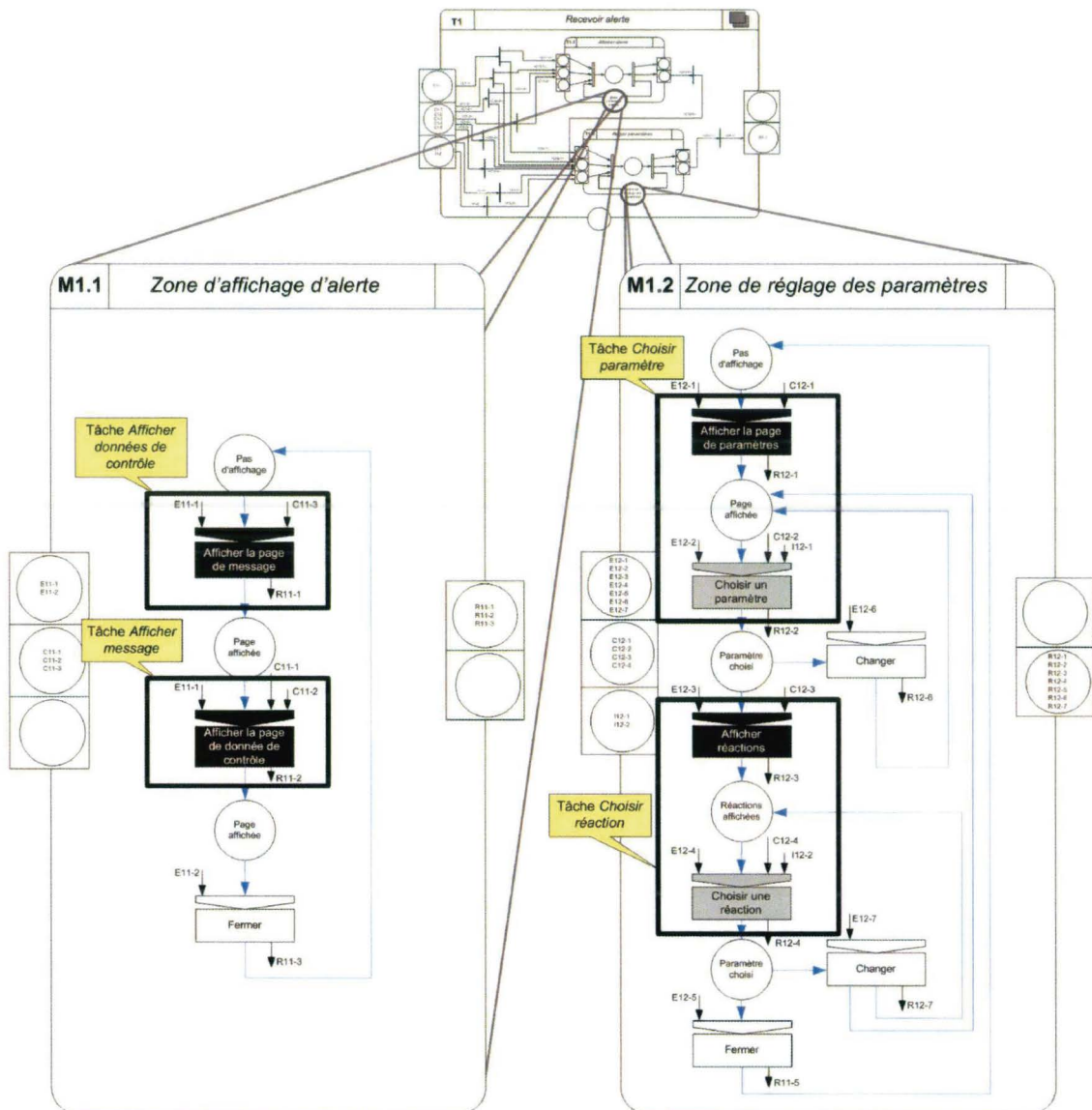


Figure IV.88 : Deux zones d'interaction pour les deux sous-tâches T1.1 et T1.2

Un vocabulaire est choisi pour pouvoir faire le lien entre les éléments constituant le modèle de tâche et les modèles d'IHM de TOOD, et ceux définis par notre méthode. Du point de vue du vocabulaire de conversion, trois types de transitions peuvent être distingués dans la zone d'interaction TOOD :

- Transition automatique (en noir) : correspondante à une action qui se déroule automatiquement par le système au travers de l'IHM ; par exemple, affichage d'une liste à l'utilisateur.
- Transition d'interaction (en gris) : est franchi directement lors d'une action de la part de l'utilisateur ; par exemple action de choix d'un item dans une liste. Cette transition est éventuellement accompagnée d'une transition automatique.

- Transition interne (en blanc) : d'une part, ce type de transition peut être utilisé par le noyau fonctionnel des composants ; par exemple, action de mise à jour d'une liste d'items ; d'autre part ce type de transition peut être utilisé directement par le noyau fonctionnel du système afin d'achever les tâches système ; par exemple la tâche de quitter le système interactif.

Nous commençons par construire le modèle AMXML de notre méthode, à partir du modèle de tâche TOOD. Rappelons que le modèle de tâche de TOOD comprend une tâche principale *Recevoir alerte* et deux sous-tâches *Afficher alerte* et *Régler paramètres*.

Dans le cadre de la construction du modèle AMXML, nous partons de l'hypothèse que pour chaque transition d'interaction ou un groupe de deux transitions automatique et d'interaction dans la zone d'interaction de TOOD, une tâche interactive est ajoutée à l'AMXML, et reliée directement à sa tâche mère. A partir de la zone d'interaction de la première sous-tâche *Afficher alerte* (Figure IV.88, partie gauche), deux transitions d'interaction sont distinguées (encadrées en noir sur la figure) : la première, *Afficher donnée de contrôle*, permet de présenter la valeur atteinte pour la donnée de contrôle concernée dans l'alerte ; et la deuxième, *Afficher message*, permettant de montrer à l'opérateur un éventuel message de l'alerte reçue. Ces deux tâches interactives sont en concurrence (suite au lien direct entre les deux transitions dans la zone d'interaction de TOOD).

Ensuite, dans la zone d'interaction de la deuxième sous-tâche *Régler paramètres* (Figure IV.88, partie droite), deux groupes de transitions automatique/d'interaction sont distinguées (encadrés en noir sur la figure) : le premier groupe *Choisir paramètre* permet de présenter à l'utilisateur une liste de paramètres à régler (il s'agit d'en sélectionner un), et le deuxième *Choisir réaction* qui permet d'afficher une liste des valeurs possibles pour le paramètre choisi, en laissant à l'utilisateur d'en choisir une. Les deux tâches interactives ajoutées à l'AMXML sont en activation avec échange de données.

Afin de faciliter au lecteur ne connaissant pas TOOD la compréhension de la hiérarchie des tâches concernées, toutes les tâches sont représentées en CTT (cf. Figure IV.89).

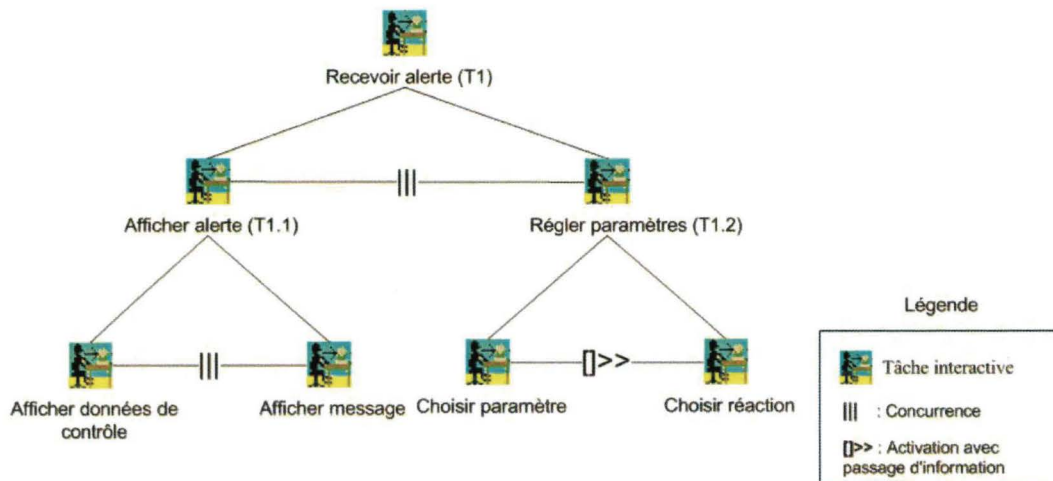
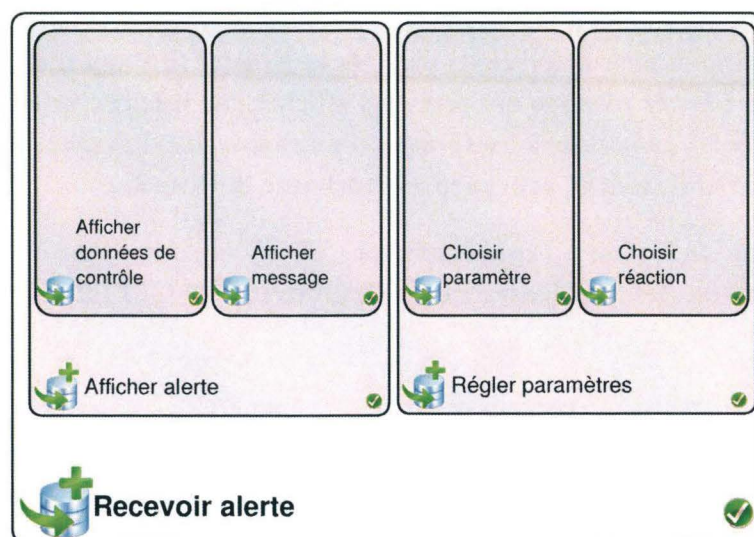


Figure IV.89 : Traduction en CTT du modèle de tâche "Recevoir alerte" TOOD

Afin de générer le modèle AMXML (comme le préconise notre méthode), les règles de génération d'AMXML (voir chapitre 3, §III.3) sont appliquées. Tout d'abord, nous appliquons la règle 2 sur le modèle de tâche : par conséquent, des *conteneurs abstraits* sont associés à la tâche principale *Recevoir alerte*, et aux sous-tâches mères *Afficher alerte*, *Régler paramètres*. Ensuite des *composants abstraits* sont associés aux tâches interactives terminales *Afficher donnée de contrôle* et *Afficher message* de la tâche mère *Afficher alerte* ; et des *composants abstraits* sont associés aux tâches terminales *Choisir paramètre* et *Choisir réaction* de la tâche mère *Régler paramètres*. Les *composants abstraits* sont placés dans les *conteneurs abstraits* de leurs tâches mères. La Figure IV.90 illustre les *conteneurs abstraits* et les *composants abstraits* associés aux tâches et représentés en UsiXML. Les *conteneurs abstraits* et les *composants abstraits* sont organisés en respectant la hiérarchie des tâches.

La partie `<taskModel>` de l'AMXML se compose des tâches du système et d'une sous-partie `<relationships>` permettant de spécifier les opérateurs entre les tâches. Tout d'abord des balises `<task>` sont associées à la tâche mère *Recevoir alerte*, aux sous-tâches mères *Afficher alerte*, *Régler paramètres*, et aux quatre tâches interactives terminales *Afficher donnée de contrôle*, *Afficher message*, *Choisir paramètre*, et *Choisir réaction*. Ensuite les opérateurs entre les tâches (activation, concurrence,...) sont intégrés dans la partie `<relationships>`. Une balise `<link>` est associée à chaque opérateur, en spécifiant les deux tâches concernées par l'opérateur dans les sous-balises `<source>` et `<target>`.



```

<uiModel>
<abstractContainer id="c_recevoir_alerte" name="Recevoir alerte">
<abstractContainer id="c_afficher_alerte" name="Afficher alerte">
<abstractComponent id="com_afficher_donnes" name="Afficher données de contrôle"/>
<abstractComponent id="com_afficher_message" name="Afficher message"/>
</abstractContainer>
<abstractContainer id="c_regler_parametres" name="Régler paramètres">
<abstractComponent id="com_choisir_parametre" name="Choisir paramètre"/>
<abstractComponent id="com_choisir_reaction" name="Choisir réaction"/>
</abstractContainer>
</abstractContainer>
</uiModel>

```


Figure IV.90 : Modèle d'interface abstraite de la tâche "Recevoir alerte" en UsiXML

Dans la partie `<context>`, le concepteur peut spécifier un contexte dans lequel un utilisateur ayant une autorisation de supervision de niveau « 3 » est défini. Selon le niveau d'autorisation, un filtrage des paramètres peut être effectué. Suite à ce filtrage, l'opérateur humain concerné pourra modifier des paramètres, ou encore les valeurs qu'il/elle peut donner à un paramètre, selon son profil. Ensuite, le concepteur définit les ressources textuelles de chaque *composant abstrait* ou *conteneur abstrait* par rapport au contexte prédéfini. Ces ressources seront utilisées pour la génération de l'IHM finale, dans la prochaine étape.

La Figure IV.91 illustre le modèle AMXML de la tâche *Recevoir alerte*. Dans la partie suivante (niveau concret), la structure métier du système de supervision sera construite à base de composants métier.

```

<amxml>
  <taskModel> // Définir les tâches du système
    <task id="t_recevoir_alerte" name="Recevoir alerte" type="interactive" container="c_recevoir_alerte">
      <task id="t_afficher_alerte" name="Afficher alerte" type="interactive" container="c_afficher_alerte">
        <task id="t_afficher_donnes" name="Afficher donnes" type="interactive" function="showCotrData" container="com_afficher_donnes"/>
        <task id="t_afficher_message" name="Afficher message" type="interactive" function="showMsg" container="com_afficher_message"/>
      </task>
      <task id="t_regler_parametres" name="Régler paramètres" type="interactive" container="c_regler_parametres">
        <task id="t_choisir_parametre" name="Choisir paramètre" type="interactive" function="choice" container="com_choisir_parametre"/>
        <task id="t_choisir_reaction" name="Choisir réaction" type="interactive" function="choice" container="com_choisir_reaction"/>
      </task>
    </task>
    <relationships> // Définir les liens entre les tâches
      <link id="l_recevoir_alerte_regler_parametres" type="concurrence">
        <source sourceId="t_recevoir_alerte" out="default"/><target targetId="t_regler_parametres" in="default"/></link>
      <link id="l_afficher_donnes_afficher_message" type="concurrence">
        <source sourceId="t_afficher_donnes" out="default"/><target targetId="t_afficher_message" in="default"/></link>
      <link id="l_choisir_parametre_choisir_reaction" type="enabling_data">
        <source sourceId="t_choisir_parametre" out="default"/><target targetId="choisir_reaction" in="default"/></link>
    </relationships>
  </taskModel>
  <uiModel> // Définir les conteneurs et les composants abstraits de l'IHM
    <abstractContainer id="c_recevoir_alerte" name="Recevoir alerte">
      <abstractContainer id="c_afficher_alerte" name="Afficher alerte">
        <abstractComponent id="com_afficher_donnes" name="Afficher données de contrôle"/>
        <abstractComponent id="com_afficher_message" name="Afficher message"/>
      </abstractContainer>
      <abstractContainer id="c_regler_parametres" name="Régler paramètres">
        <abstractComponent id="com_choisir_parametre" name="Choisir paramètre"/>
        <abstractComponent id="com_choisir_reaction" name="Choisir réaction"/>
      </abstractContainer>
    </abstractContainer>
  </uiModel>
  <contextModel> // Définir les contextes prévus
    <context id="co1">
      <environment id="e1" type="any" workstate="supervision"/>
      <platform id="p1" type="PC"/>
      <user id="u1" language="FR" permission="p3"/>
    </context>
  </contextModel>
  <resourceModel> // Définir les ressources des composants de l'IHM
    <resource id="r_c_recevoir_alerte_co1" component="c_recevoir_alerte" context="co1" text="Alerte reçue"/>
    <resource id="r_com_choisir_parametre_co1" component="com_choisir_parametre" context="co1" text="Choisir un paramètre à régler"/>
    <resource id="r_com_choisir_reaction_co1" component="com_choisir_reaction" context="co1" text="Régler le paramètre"/>
  </resourceModel>
</amxml>

```

Figure IV.91 : AMXML de l'IHM « Recevoir alerte »

III.3.2 Construction de la structure métier

Au niveau concret, la structure métier est construite en appliquant des règles de construction de la structure métier (cf. Chapitre III, §IV.5). Tout d'abord, en appliquant la règle 1, des composants métier "conteneur" sont associés aux *conteneurs abstraits* : *Recevoir alerte*, *Afficher alerte* et *Régler paramètres*. La règle 3 peut être appliquée sur les deux *conteneurs abstraits* *Afficher alerte* et *Régler paramètres*, puisque leurs tâches associées s'exécutent sans aucun ordre entre eux. Par conséquent, les deux précédents composants "conteneur" sont remplacés par deux composants métier "panneau".

Ensuite, la règle 5 permet d'investir l'expérience potentielle des patrons de conception à choisir des composants métier pour les quatre *composants abstraits*, en fonction du type des tâches associées. Tout d'abord des composants métier "choix" sont associés aux deux *composants abstraits* *Choisir paramètre* et *Choisir réaction*. Un composant métier "texte" est associé au *composant abstrait* *Afficher message*. Enfin, un composant métier spécifique "DonneeCtrl" est associé au *composant abstrait* *Afficher donnée de contrôle* [Hariri et al., 2007b].

Enfin, des composants métier "sortie" et "validation" sont associés au *conteneur abstrait* *Recevoir alerte*, en appliquant les règles 6 et 7. La Figure IV.92 illustre les *conteneurs* et *composants abstraits* avec leurs composants métier associés.

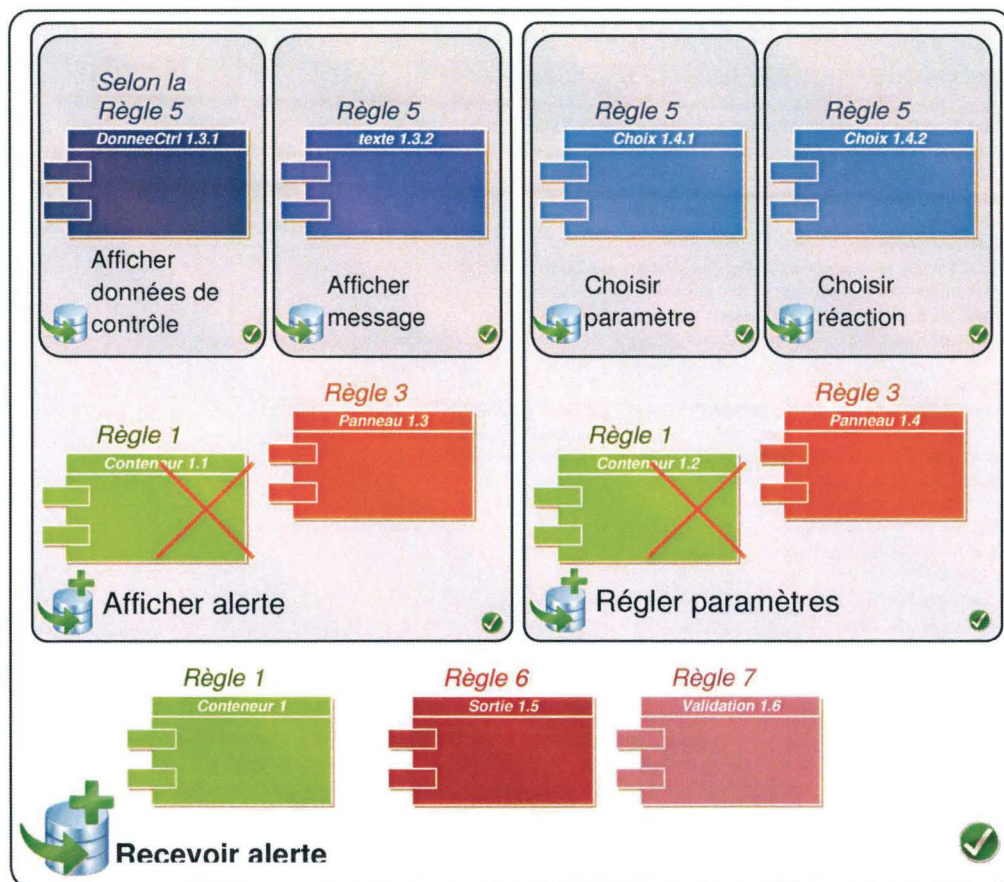


Figure IV.92 : Application des règles de génération de la structure métier

Après avoir sélectionné les composants métier pour le système de supervision, les composants sélectionnés doivent être reliés (cf. Figure IV.93) en respectant la partie *<relationships>* du modèle AMXML, et à l'aide de patrons de conception. Tout d'abord les composants métier "Donnee Ctrl" 1.3.1 et "Message" 1.3.2 sont reliés. Ensuite tous les deux sont reliés avec le composant métier "panneau" 1.3. Les composants métier "choix" 1.4.1 et 1.4.2 sont reliés. Ensuite tous les deux sont reliés avec le composant métier "panneau" 1.4. Les deux composants métier "panneau" 1.3 et 1.4, les composants métier "sortie" 1.5 et "navigation" 1.6 sont reliés avec le composant métier "conteneur" 1.

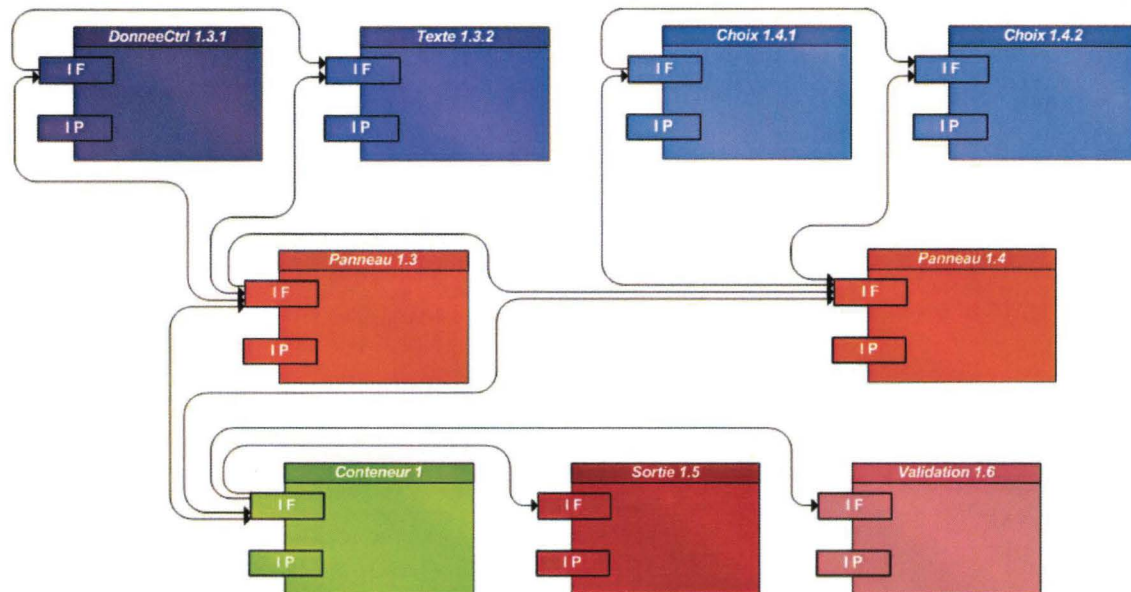


Figure IV.93 : Structure métier « Recevoir alerte »

III.3.3 Adaptation de l'IHM primaire

Nous passons à la deuxième phase du niveau concret. Dans cette étape, une première version de l'IHM plastique est générée à base des composants de présentation. Ces derniers sont sélectionnés selon le contexte d'usage prédéfini par le concepteur (utilisateur français, plate-forme de type PC, modalité graphique).

Tout d'abord, un composant de présentation de type fenêtre est choisi pour le composant métier "conteneur" 1. Deux composants de présentation de type "panneau" spécifiques aux deux composants métier "panneau" 1.3 et 1.4 sont sélectionnés. Ceux-ci sont ajoutés à la fenêtre du composant métier 1, et contiendront par la suite les composants de présentation sélectionnés pour les composants métier 1.3.1, 1.3.2, 1.4.1 et 1.4.2.

La Figure IV.94 illustre l'arbre de décision du composant "DonneeCtrl" pour décider des échelles de mesure nécessaires pour chaque type de ses composants de présentation. Le

premier composant de présentation "Cadran"²⁴ affiche une valeur de donnée mesurée en échelle intervalle (voir §III.2.2.1), avec la possibilité de montrer à l'utilisateur les différents champs de donnée de contrôle (par exemple, bas, normal, élevé) afin d'aider à prendre des décisions. Pour afficher ces champs, une transformation des données en échelle de mesure ordinale devra être nécessaire. Ensuite, le composant de présentation "Barre" permet d'afficher la valeur dans une barre (échelle intervalle). La couleur de la barre change selon le champ de la valeur reçue (échelle nominale). Il affiche également, en dessus de la barre, les différents champs de donnée de contrôle (échelle ordinale). Enfin le composant "texte", adéquat à un petit espace d'affichage, affiche la donnée de contrôle au format texte (échelle intervalle). L'échelle de mesure nominale permet de spécifier la couleur du texte afin d'indiquer : si la valeur reçue est citée au niveau normal (couleur noir), ou a atteint une valeur critique (couleur rouge).

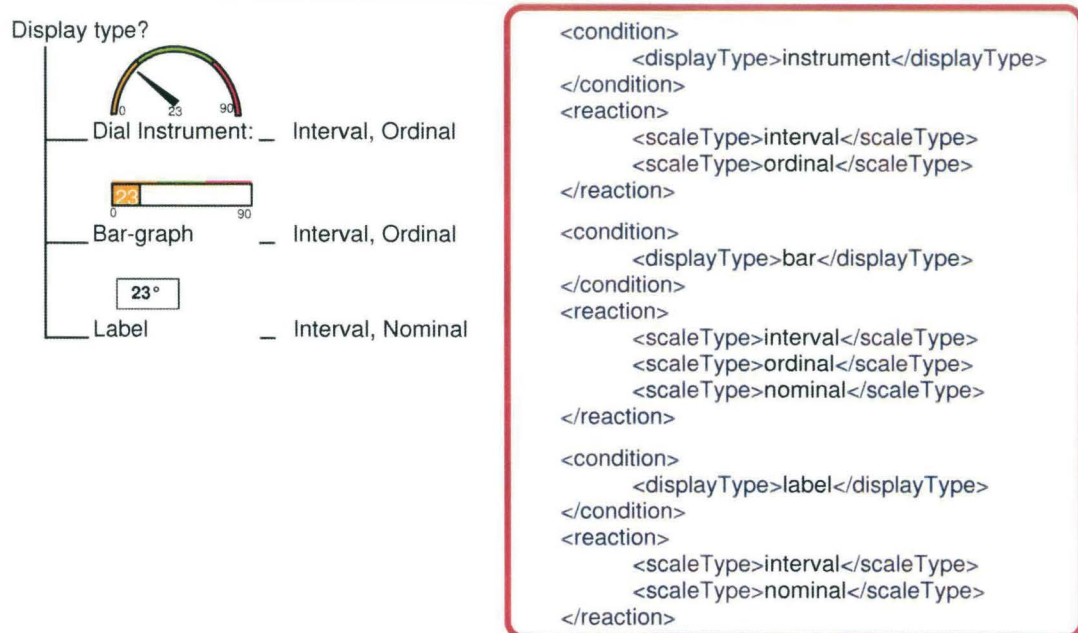


Figure IV.94 : Arbres de décision du composant "DonneeCtrl" pour décider des échelles de mesure nécessaires

Le composant métier "DonneeCtrl" 1.3.1 choisit un composant de présentation de type "Cadran". Le composant de présentation "Cadran" occupe normalement un espace d'affichage important, mais il n'y a pas de conflit à ce sujet dans notre cas (plate-forme de type PC, l'espace d'affichage s'avérant suffisant). Ensuite le composant métier "texte" 1.3.2 choisit une boîte de texte dans laquelle le message reçu peut être affiché.

²⁴ Un composant de présentation de type "Cadran" (traduit en anglais par : *dial*) permet d'afficher une valeur dans un instrument d'indication sous forme de cadran. Il est équipé d'un pointeur qui indique la valeur avec une variété d'informations qui indiquent par exemple le niveau normal et celui d'urgence d'une valeur.

Comme l'espace d'affichage envisagé est suffisant, les composants métier "choix" 1.4.1 et 1.4.2 choisissent des composants de présentation de type "ListBox". Ensuite, les composants "sortie" et "validation" choisissent deux composants de type bouton, un pour quitter le système et l'autre pour valider le choix et/ou la saisie de l'utilisateur.

Les composants de présentation sont arrangés sur l'IHM grâce aux patrons de conception. Ces derniers arrangent horizontalement les deux panneaux ; ensuite leurs composants de présentation sont verticalement arrangés. Puis le système est équipé des processus d'apprentissage et de capture du contexte. A la fin de cette étape, l'IHM finale est prête à être distribuée sur la plate-forme cible (PC) et à être exploitée par l'utilisateur final (cf. Figure IV.95).

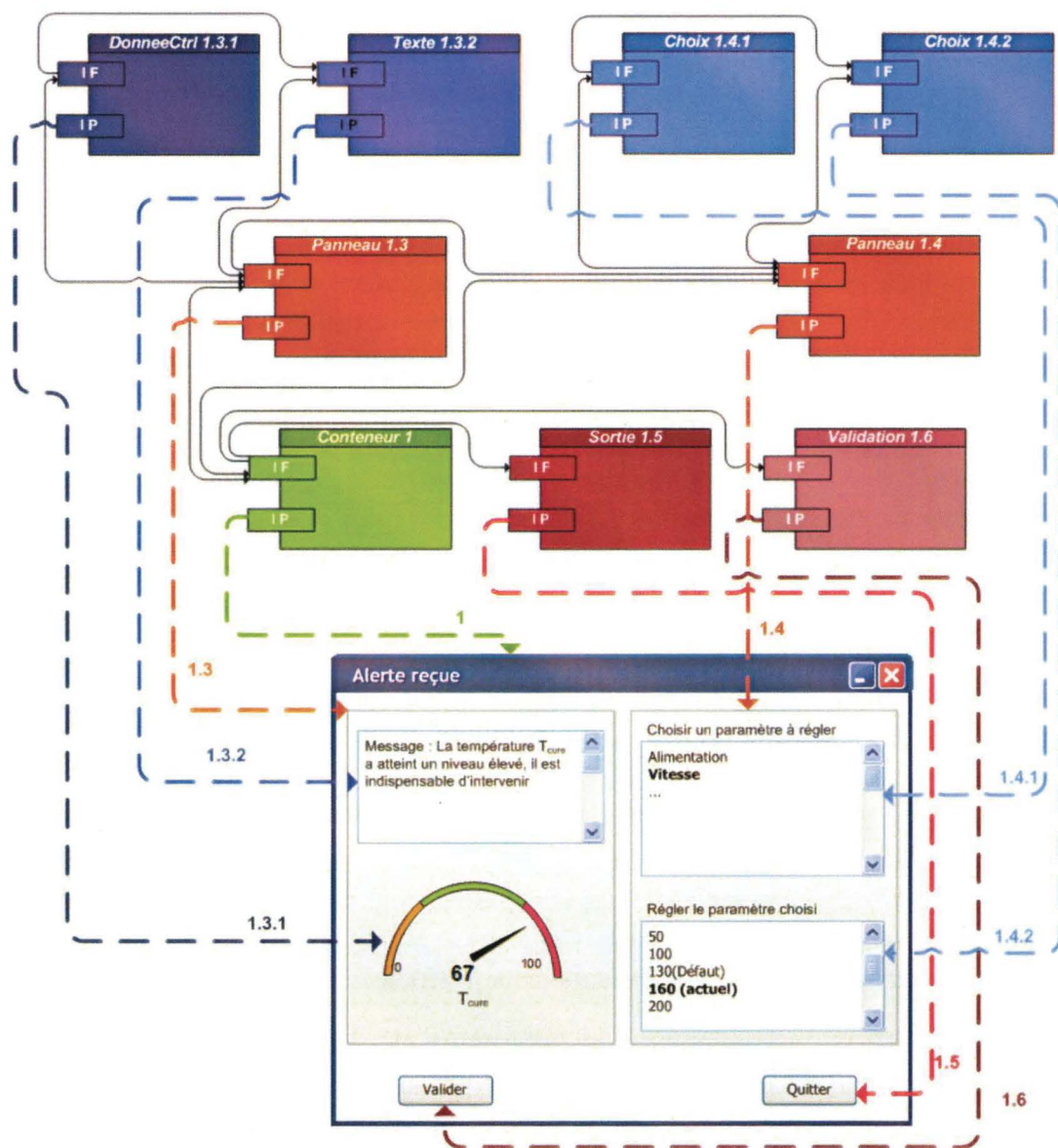


Figure IV.95 : Première version du système de supervision industrielle « Recevoir alerte »

III.3.4 Utilisation du système interactif plastique

Après la distribution du système de supervision sur le dispositif cible, l'utilisateur final commence à l'utiliser. Suite à une hausse de température T_{cure} (rappelons que la température de la cure T_{cure} ne doit pas dépasser les 55°C, voir §III.1), l'IHM générée dans la précédente partie s'affiche automatiquement à l'utilisateur (de profil : opérateur de supervision dans la salle de contrôle). Ce dernier commence à interagir avec le système, suite à l'apparition de l'alerte à traiter en urgence. En même temps, grâce à des équipements de capture du contexte, le contexte d'usage dans le système est actualisé avec des informations capturées concernant l'utilisateur, la plate-forme et l'environnement. Le processus d'apprentissage peut déduire des réactions répétées de l'utilisateur suite à des éventuels changements contextuels et de l'ajouter à la base de connaissance (cf. Chapitre III, § VI.3).

III.3.5 Adaptation vivante

L'opérateur qui a reçu le message d'alerte, quitte la salle pour vérifier, sur place, l'état de la machine. Pour cela, il/elle transfère son travail sur un PDA. Ce dernier est connecté au réseau. Ensuite le terminal de l'utilisateur détecte le PDA, et l'utilisateur fait migrer l'IHM vers la nouvelle plate-forme. Suite à l'important changement au niveau du contexte lié à la plate-forme, l'adaptation vivante appelle les composants métier à changer leur facette de présentation en adéquation avec les nouvelles caractéristiques de la plate-forme (PDA, possédant un espace d'affichage limité de 360x240). Les composants métier "*Choix*" 1.4.1 et 1.4.2, remplacent les composants de présentation "*ListBox*" par ceux de type "*ComboBox*" (ces types de composants sont déjà utilisés dans le chapitre III). Les composants métier "*Validation*" et "*Sortie*" sont intégrés dans la barre en-dessous de l'écran.

Enfin, le composant métier "*DonneeCtrl*" choisit un composant de présentation de type "*Barre*". Celui-ci nécessite un petit espace d'affichage pour une valeur de la donnée de contrôle. La température est transformée en échelle de mesure ordinale (voir §III.2.2.1). Cette transformation est nécessaire pour détecter le champ actuel de la température (dans notre cas, la température T_{cure} est trop élevée : elle est donc affichée en rouge, en cohérence avec la norme N.F. X. 08-003-1, cf. [AFNOR, 2006]). La couleur de la barre est modifiée selon le champ indiqué (cf. Figure IV.96, partie gauche). Ensuite le patron "*arrangeur*" effectue l'arrangement des composants de présentation en utilisant une autre stratégie d'arrangement des panneaux. Cette stratégie permet de placer verticalement les panneaux en répondant au changement du mode d'affichage (mode d'affichage vertical).

On suppose que l'utilisateur n'a pas pu intervenir à cause d'un manque de connaissance profonde relativement au problème qui est inhabituel. C'est pourquoi, l'alerte est transférée à un opérateur d'astreinte (expert du domaine). Ce dernier est équipé d'un téléphone portable possédant un écran d'affichage 320x240. Le processus d'adaptation vivante est relancé. Le composant métier "*DonneeCtrl*" choisit de nouveau un composant de présentation de type "*Texte*". Celui-ci nécessite un très petit espace d'affichage. Il affichera la température T_{cure} en couleur rouge (car elle a dépassé 55°C) et les limites du

champ normal en vert. Ensuite les deux cadres sur l'IHM correspondant aux facettes de présentation des composants métier "panneau" 1.3 et 1.4 sont enlevés (en respectant le guide de style du téléphone portable). La liste des paramètres et des réactions est mis à jour suite au changement du profil de l'utilisateur (en particulier le niveau d'autorisations) (cf. Figure IV.96, partie droite). Enfin, le patron "Régleur de taille et de localisation" recalcule la taille et la localisation considérées comme optimales pour les composants de présentation.



Figure IV.96 : IHM adaptée pour un PDA, régénération pour un téléphone portable

III.3.6 Processus d'apprentissage

L'opérateur d'astreinte ayant reçu l'alerte sur son téléphone portable, a besoin de vérifier les valeurs limites (haute et basse) de température de la cuve (cas de situation normale), pour intervenir. Comme l'IHM est équipée de mécanismes donnant la possibilité à l'utilisateur de changer un composant, l'opérateur a demandé de changer le composant "texte" qui est le responsable de l'affichage de la température T_{cure} , en choisissant l'affichage des valeurs limites (cf. Figure IV.97). Le texte ajouté aura la couleur prédéfinie pour le niveau normal (on suppose que la couleur verte a été choisie, en cohérence avec la norme N.F. X. 08-003-1).



Figure IV.97 : IHM modifiée par l'utilisateur

Le processus d'apprentissage prend en compte cette réaction de l'utilisateur (plate-forme : *téléphone*, utilisateur : *expert*, composant métier : *donneeCtrl*, composant de présentation : *texte*, réaction : *ajout des limitations du niveau normal*). A partir de cet exemple capturé, le processus d'apprentissage met à jour la base de connaissance du composant métier "DonneeCtrl", en ajoutant à la partie concernant les échelles de mesure, l'échelle ordinale pour le composant de présentation "texte" (cf. Figure IV.98).

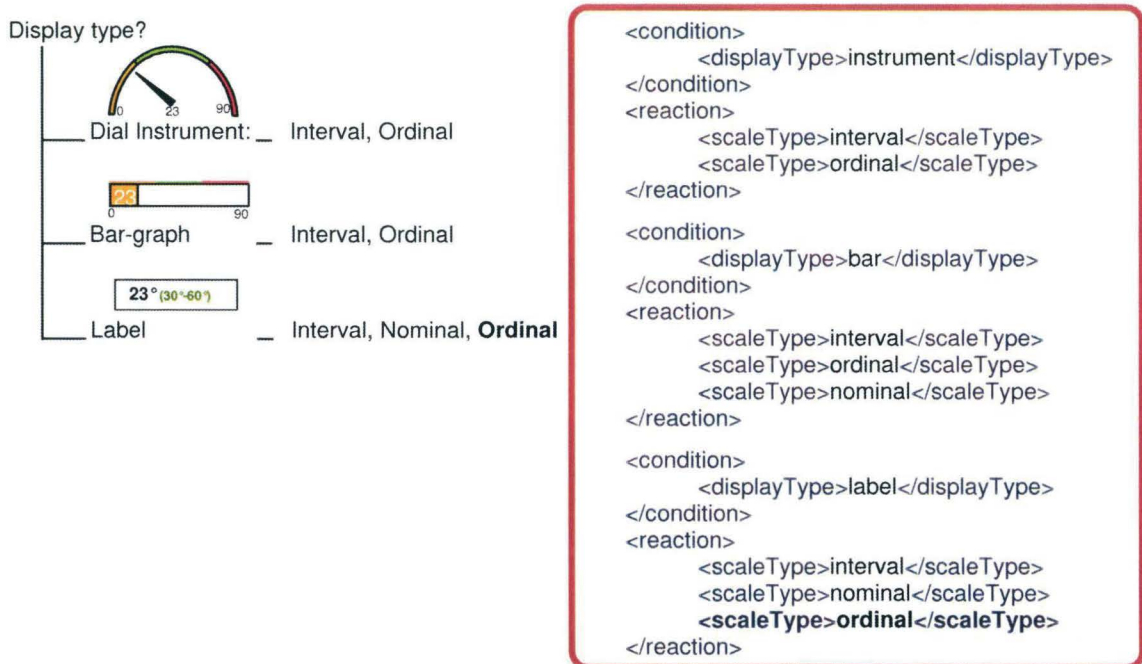


Figure IV.98 : Arbre de décision du composant "DonneeCtrl" amélioré

Ensuite, la nouvelle spécification du composant métier (par l'utilisateur) est stockée dans les préférences de l'utilisateur (afin de la considérer lors d'une future utilisation du système par l'utilisateur). L'expert du système a l'autorisation d'accéder aux préférences des utilisateurs ; lorsqu'il trouve pertinentes des nouvelles connaissances déduites pour une certaine catégorie d'utilisateur et dans des contextes d'usages susceptibles de se répéter, il peut valider et adopter des nouvelles connaissances et les ajouter à la base centrale de connaissance du système.

III.4 Conclusion sur le système de supervision d'une usine chimique

Nous avons étudié dans le cadre de cette seconde étude de cas une mise en œuvre possible d'application industrielle (simulée) en suivant les étapes de notre méthode de génération d'IHM plastique. Nous nous sommes cette fois basés sur l'hypothèse que la partie du système sur laquelle l'application de la méthode a été effectuée, était déjà spécifiée dans la méthode TOOD. Les tâches ainsi que des spécifications concernant le système ont été intégrées dans notre modèle AMXML. Ensuite un modèle AMXML a été généré. Le concepteur a enrichi ce modèle en spécifiant ses parties *<context>* et *<resources>*. La structure métier du système a été construite à base des composants métier. Un nouveau type de composant a été utilisé pour afficher des données de contrôle dans le système de supervision industriel. Le composant permet de présenter une donnée de contrôle de différentes manières selon l'espace d'affichage disponible. De plus ce composant utilise la notion d'échelle de mesure visant à convertir la valeur reçue adéquatement au type d'affichage. Une première version du système a été générée pour un opérateur humain présent dans un premier temps en salle de contrôle. L'IHM s'est adaptée deux fois suite à deux migrations vers de nouvelles plateformes. Pendant l'adaptation, des composants de présentation ont été changés, et différentes stratégies d'arrangement des composants sur l'IHM ont été appliquées.

IV. Conclusion

Ce chapitre a été scindé en deux parties.

La première a été consacrée à une application dans le domaine touristique. Un scénario ainsi qu'un modèle de tâche d'un système de guidage touristique a été expliqué. Ensuite la méthode proposée a été appliquée étape par étape sur une partie du système de guidage en suivant les étapes de génération et d'adaptation des IHM. Suivant des réactions répétées par une certaine catégorie de l'utilisateur, des nouvelles connaissances ont été ajoutées au système, par le processus d'apprentissage. Nous avons vu la nécessité de l'adaptation vivante avec la prise en compte des nouvelles connaissances. Autrement, l'utilité de l'adaptation vivante aurait été perdue progressivement.

Un système de supervision industrielle a fait l'objet de la deuxième partie. L'étude de ce système a suivi aussi les différentes étapes de notre méthode. Nous avons commencé par le scénario d'un système de supervision dans une usine de remplissage de godets, avec des spécifications et des contraintes de production. Les différentes facettes du contexte

d'usage ont été présentées. Dans le domaine de la supervision, l'importance des données de contrôle et de leur présentation sur l'IHM, a introduit la notion d'échelle de mesure et la transformation entre les différentes échelles de mesure, dans notre méthode. S'appuyant sur un vocabulaire de conversion, des spécifications du système de supervision en TOOD ont été représentées et intégrées dans le modèle AMXML. Ce lien entre TOOD et notre méthode assure un des objectifs de notre méthode, qui est celui de montrer une connexion possible entre les méthodes classiques de spécification et conception des systèmes interactifs et la notion de plasticité.

Une discussion sur les résultats faisant suite aux études de cas, et les perspectives de recherche font l'objet du dernier chapitre.

Chapitre V

Discussion et perspectives de recherches

I. Introduction

Dans ce dernier chapitre, nous allons d'abord proposer une discussion sur l'originalité de notre contribution. Nous allons mettre en évidence les avantages et les inconvénients de la méthode proposée, par rapport aux principes de conception et d'implémentation d'IHM plastique, tout en prenant en considération les résultats faisant suite aux deux études de cas illustrées dans le précédent chapitre.

Ensuite, la deuxième partie sera consacrée à la présentation de nos perspectives de recherche et de développement à court, moyen et long termes.

II. Originalité de la méthode proposée

L'originalité de la méthode proposée dans le cadre de la thèse, tient aux principes sur lesquels elle s'articule. Ces principes sont tous motivés par des besoins récurrents en conception et implémentation d'IHM plastique. Nous avons centré notre réflexion sur les mécanismes permettant de préserver l'utilisabilité de l'IHM.

L'originalité de notre contribution se situe aussi bien au niveau conceptuel de l'IHM, qu'à celui de l'adaptation.

Sur le plan conceptuel, nous proposons une méthode en quatre étapes servant le processus de conception d'IHM plastique, cette méthode s'appuyant sur cinq principes (qui seront discutés dans § III.1 et §III.3) :

- Le principe de conception d'IHM sur lequel nos contributions se basent, favorise la capitalisation et la réutilisation de connaissances de conception.
- La réification (traduction) des spécifications abstraites d'IHM d'autres méthodes classiques, orientées multicibles, ou orientées plasticité dans notre méthode, permet de faire le lien avec ces autres méthodes.
- L'utilisation des composants métier qui favorise la capitalisation et la réutilisation des composants dans l'architecture du système interactif plastique, de même que la minimisation des efforts de développement et de maintenance.
- L'architecture proposée pour les systèmes interactifs plastiques fait un découpage clair entre les différentes couches du système.
- Le développement des connaissances de conception à l'exécution grâce à des techniques d'apprentissage, permet de minimiser les efforts de gestion et maintien des connaissances.

Sur le plan de l'adaptation, nous avons proposé deux principes (qui seront discutés dans §III.2) :

- Les principes de choix et changement des composants de présentation qui favorisent l'adaptation de l'IHM au contexte d'usage prédéfini et ensuite aux contextes changés.
- Le principe de classification des règles d'adaptation selon le contexte permet de faciliter la réalisation de l'adaptation, de même que la minimalisation des efforts de développement et de maintien.

III. Points forts

Nous allons dans cette section aborder les points forts de notre méthode en rapport avec la problématique des IHM plastiques.

III.1 Spécification et conception d'IHM plastique

La conception de l'IHM plastique est l'un des points centraux de la méthode proposée. La méthode suit les étapes du cadre Cameleon (cf. chapitre II, §III.8). Elle s'appuie sur une architecture du système interactif plastique à base de composants métier. Ceux-ci possèdent des composants fonctionnels qui contribuent au noyau fonctionnel du système, des composants de présentation qui peuvent être utilisés pour construire l'IHM, et une base de connaissance supportant la prise de décision quant aux choix de présentation.

Grâce aux vocabulaires prédéfinis, notre méthode vise à générer les IHM à partir de spécifications d'IHM venant d'autres méthodes et/ou d'un modèle de tâches. Le modèle de tâches et les spécifications peuvent être converties et représentées au format adaptable à notre méthode (Modèle AMXML). Le modèle AMXML a été proposé comme une extension du MIA d'UsiXML. Ainsi, dans la première étude de cas, la méthode a démarré à partir d'un modèle d'IHM en UsiXML, alors que dans la deuxième étude de cas, un modèle d'IHM en TOOD a fait office de point de départ.

Les connaissances de conception sont capitalisées et représentées sous forme des patrons de conception. La méthode s'appuie sur deux bibliothèques de patrons : la première est destinée aux choix et aux assemblages des composants métier, alors que la seconde permet d'harmoniser et d'arranger les composants de présentation sur l'IHM. Notons que la méthode MOBI-D (cf. Chapitre II, §III.8) s'appuie sur une base de connaissances (guides de style, patrons de conception...) à propos de directives et de conseils opérationnels pour la conception d'interface, cependant MOBI-D ne propose aucun outil pour le développement des patrons de conception.

L'architecture du système interactif plastique est organisée en trois couches : (1) couche de présentation (où toutes les facettes de présentation sont accessibles), (2) couche fonctionnelle (dans laquelle la structure métier est située, de même que les processus d'apprentissage et de capture du contexte d'usage), (3) couche XML au niveau de laquelle les bibliothèques de patrons de conception, le guide de style, les règles d'adaptation, et le modèle AMXML sont installés. A ce sujet, le projet Rainbow (cf. Chapitre II, § III.6) expose une architecture dédiée aux systèmes auto-adaptables. Il est intéressant de nous

positionner par rapport à cette architecture se composant de trois couches : (1) la couche d'architecture où des informations sont capturées et évaluées afin de détecter le besoin d'une adaptation, la machine d'adaptation se situant dans cette couche, cette couche correspondant à la couche fonctionnelle de notre architecture ; (2) la couche de traduction qui établit la correspondance entre la couche d'architecture et celle du système d'administration, cette couche correspondant aux sous-couches pouvant se trouver entre les trois couches de notre architecture (elles n'ont pas été détaillées dans ce mémoire) ; (3) et la couche système où les interfaces sont implantées et où des capteurs observent et mesurent les différents états du système, cette couche correspondant à la couche de présentation de notre architecture.

[Dâassi *et al.*, 2003] ont proposé une définition, une modélisation et une architecture logicielle permettant la composition de *Comets* (cf. chapitre II, § III.10) au sein de systèmes interactifs. [Demeure, 2007] a continué à travailler sur la notion de *Comet*. Il a proposé un modèle appelé *Ecosystème* permettant de décrire un système interactif, son contexte d'usage et le déploiement du système interactif dans son contexte d'usage. *Ecosystème* intègre des descriptions à tout niveau d'abstraction et permet de raisonner sur le système interactif "en contexte".

Notre contribution intègre des spécifications du contexte d'usage au niveau abstrait de la méthode ; ensuite le contexte est pris en compte, au niveau concret, par l'adaptation primaire. Enfin à l'exécution, l'adaptation vivante peut prendre en compte les changements contextuels pendant l'exécution.

III.2 Adaptation

Nous avons adopté le format « Action * Condition → Réaction » [Calvary *et al.*, 2003] pour spécifier les règles d'adaptation. Celles-ci sont initialement définies à la conception par le concepteur. Ensuite, les règles peuvent être améliorées, ou de nouvelles règles peuvent être ajoutées, à l'exécution.

L'adaptation peut s'effectuer premièrement à la conception (adaptation primaire), avant la distribution de l'IHM sur le dispositif cible, en prenant en compte un contexte d'usage prédéfini. De plus, l'adaptation peut se dérouler à l'exécution (adaptation vivante), suite à un nouveau contexte, sans nécessité, donc, de retourner à l'étape de conception. En effet, les informations contextuelles sont actualisées lors de la capture du contexte. Celles-ci sont envoyées régulièrement au système pour évaluer l'utilisabilité de l'IHM. En cas de changement au niveau du contexte, une adaptation est lancée. Les méthodes et langages présentés et comparés au Chapitre II, se focalisent sur l'adaptation pré-calculée et prédéfinie par le concepteur. De plus, l'adaptation est appliquée à la dernière phase de génération d'IHM et avant la distribution de l'IHM. Ensuite lors d'un changement de contexte ou de plate-forme (dans le cas des approches orientées multiplateforme), une nécessité de transférer l'IHM à la phase de conception est donc envisagée.

Nous avons centré notre réflexion sur l'adaptation vivante en laissant au second plan, sans toutefois l'ignorer, l'adaptation à la conception qui est indispensable en cas d'un

changement contextuel important nécessitant une génération de l'IHM en totalité. L'architecture du système interactif plastique se base sur des composants métier « intelligents ». L'utilisation de ce type des composants vise à faciliter et augmenter la capacité de l'adaptation dynamique au contexte d'usage. Les composants de présentation (encapsulés dans les composants métier) sont sélectionnés en cohérence avec la plateforme cible, en considérant les caractéristiques de l'utilisateur et de l'environnement d'utilisation. L'approche la plus proche de nos contributions, est celle de la *Comet* qui suppose d'embarquer plusieurs présentations décrites, en conception, à tout niveau d'abstraction ; ensuite à l'exécution le choix de stratégies et politiques d'adaptation peut s'effectuer, mais aucun mécanisme d'apprentissage permettant d'améliorer les connaissances prédéfinies n'est actuellement positionné dans une *Comet*.

Les solutions proposées lors de l'adaptation sont décrites sous forme de patrons. Ceux-ci sont décrits suivant un gabarit d'une façon permettant de sectionner des patrons selon le problème (c'est-à-dire selon les caractéristiques du nouveau contexte). En effet, les patrons sont classifiés selon leur contexte d'utilisation en trois catégories : patrons à appeler à la conception, ceux à utiliser à l'exécution, et ceux qui peuvent être appliqués à la conception et à l'exécution. A l'exécution, l'adaptation peut s'effectuer totalement sans nécessité de retourner à l'étape de conception sauf en cas de nécessité de reconstruction de l'IHM en totalité. [Demeure, 2007] dans sa thèse considère le système interactif comme "un graphe de *Comets*" sur lequel des transformations sont applicables ; à ce sujet il propose un annuaire de systèmes interactifs appelé GDD (Graphe Des Descriptions) dans lequel la *Comet* peut récupérer des présentations sur mesure. Ainsi, un langage de style dédié à la plasticité, et appelé CSS++, a été proposé.

III.3 Apprentissage

Nos contributions dans cette thèse visent à supporter l'IHM plastique avec une capacité d'apprentissage. L'intégration de la notion d'apprentissage permet d'améliorer les règles de décision potentielles dans les arbres de décision (format adapté par notre méthode). Nous avons adopté l'algorithme C4.5 qui est un algorithme de classification utilisé dans le domaine de l'apprentissage supervisé pour construire l'arbre de décision à partir d'exemples. L'apprentissage s'effectue donc à partir d'exemples (réactions de l'utilisateur observées et changements contextuels détectés) en comparant le changement au niveau contextuel et les réactions répétées de l'utilisateur. De nouvelles règles d'adaptation peuvent être apprises (à appliquer par la suite dans des cas semblables), ou des règles existantes déjà dans les bases de connaissance peuvent être modifiées. Avec les nouvelles connaissances, nous pouvons mettre à jour les bases de connaissance connectées aux composants métier actuels du système, comme nous l'avons vu dans la deuxième étude de cas (cf. chapitre IV, §III.3.5) ; ou celles stockées dans les bibliothèques de composants métier et de patrons de conception, comme nous l'avons vu dans la première étude de cas (cf. chapitre IV, §II.4.4).

L'accès à la base de connaissance et les modifications au niveau des règles seront aisément effectués grâce à la représentation de l'arbre de décision (construit en utilisant l'algorithme C4.5) en format standard de type XML.

Les méthodes et langages présentés au Chapitre II ne supportent pas clairement l'amélioration de la qualité de l'adaptation (la plupart d'eux se basent sur une adaptation pré-calculée).

IV. Limites

Notre thèse a un certain nombre de limites. Celles-ci tiennent principalement à la mise en œuvre. En effet, la méthode proposée n'est pas supportée par un environnement global de développement ; aucun outil de développement complet n'a non plus été développé pour la création et la gestion des patrons de conception et métier.

Un ou plusieurs patrons de conception peuvent être développés pour un problème particulier ; ce principe peut conduire à un nombre de patrons très (trop) élevé ; cependant la cohérence entre les patrons n'était pas avancée dans le cadre de cette thèse. De même, la réutilisation des patrons déjà développés dans d'autres méthodes n'a pas été particulièrement envisagée. La notion d'anti-patron²⁵ est absente dans notre thèse. Les anti-patrons peuvent jouer un rôle complémentaire du processus d'apprentissage en permettant d'améliorer la qualité des solutions conceptuelles proposées par les patrons. Pour exécuter les patrons, un interpréteur devrait être utilisé, alors que le rôle d'un tel interpréteur n'a pas été étudié dans nos travaux.

[Budinsky et al. 1996] proposent un interpréteur (cf. Figure V.1) permettant de générer automatiquement le code exécutable des patrons. Les patrons sont représentés en se basant sur le formalisme de [Gamma *et al.*, 1995] ; chaque section (intention, motivation, etc.) est affichée dans une page séparée. L'utilisateur expert peut personnaliser l'implémentation du patron, et contrôler les paramètres globaux qui s'appliquent à tous les patrons. L'interpréteur appelé COGENT (COde GENeration Template) est invoqué par des scripts Perl. Il est composé de trois parties :

- *Présentateur (Presenter)* qui implémente l'interface utilisateur ; les patrons sont présentés à l'utilisateur d'une façon textuelle sous forme de page HTML.
- *Mapper* est un interpréteur *Perl* (les descriptions du *mapping* sont des scripts *Perl*) ; il apporte une aide dans la spécification de la coopération entre l'interface utilisateur et le générateur de code.

²⁵ Les anti-patrons décrivent les défauts de conception à éviter, la mauvaise pratique des patrons, et les défauts liés à l'utilisation abusive des patrons de conception (cf. Chapitre I, §III.1.1).

- *Générateur de code (Code generator)* qui interprète le code du patron prenant en compte les spécifications de l'utilisateur provenant de la partie *présentateur*.

Malgré que la méthode ait été appliquée sur deux cas d'études différents (chacun ayant son propre contexte d'usage), nous n'avons pas étudié de migration d'IHM vers une autre modalité totalement différente (par exemple une migration d'une plate-forme graphique à une plate-forme vocale). De plus des composants métier simples étaient utilisés pour la construction de la structure métier des systèmes : le cas de composants composites métier n'a pas été étudié.

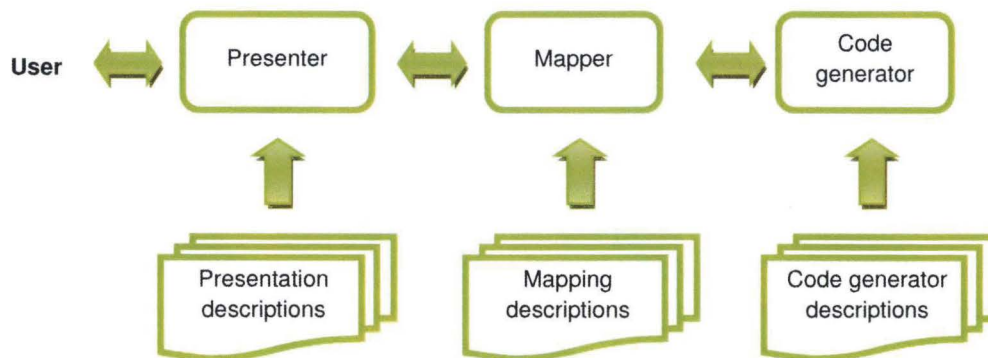


Figure V.1 : Interpréteur de patron de conception COGENT [Budinsky et al. 1996]

Des problèmes de ressources auraient pu être envisagés suite à la distribution de la bibliothèque des patrons avec le système interactif plastique, et à l'encapsulation des composants de présentation et la connaissance de sélection à l'intérieur des composants métier. Ce genre de problèmes peut limiter l'implémentation du système interactif plastique à certaines plateformes. Par exemple, un système interactif plastique composé de cinq composants métier, et distribuée avec une bibliothèque de patrons de conception (par exemple, une bibliothèque qui contiendrait 400 patrons), ne pourrait certainement pas être implémentée sur un téléphone portable avec une mémoire de 6 MO.

Ces lacunes justifient nos perspectives à court, moyen et long termes.

V. Perspectives de recherche

Nos perspectives de recherche, se situant aussi bien à court, moyen et long termes, sont successivement décrites.

V.1 À court terme

Nos perspectives à court terme sont plutôt d'ordre implémentationnelle pour compléter la validation de notre méthode et proposer une génération de composants métier « intelligents », et des patrons de conception orientés plasticité. Nos perspectives consistent à étudier en profondeur l'apprentissage et à adapter les processus d'apprentissage de capture de contexte aux caractéristiques de la plate-forme.

V.1.1 Perspectives relatives à une infrastructure logicielle

L'établissement d'une infrastructure logicielle supportant la démarche est indispensable à court terme. Cette infrastructure peut supporter chaque étape de notre méthode partant de la représentation du MIA et finissant avec la version finale de l'IHM plastique. Des mécanismes et outils existants (comme ceux fournis avec UsiXML) peuvent être utilisés par notre méthode pour supporter les différentes activités.

La réutilisation dans notre méthode de patrons développés dans le cadre d'autres méthodes, avec l'option d'intégration de ces patrons dans une des bibliothèques, devra être ajoutée. A ce sujet, l'intégration des patrons développés dans le cadre d'autres méthodes peut être partielle ou totale ; par exemple, nous pouvons intégrer une partie ou la totalité de la solution proposée par un patron pour un certain problème.

V.1.2 Perspectives relatives aux patrons et composants

Les patrons qui ont été proposés sont définis textuellement dans des fichiers XML. Un formalisme tel que P-Sigma [Conte *et al.*, 2001] associé à la définition d'une ontologie pourrait leur permettre d'être directement exploitables. La représentation graphique des patrons offrirait plus de facilités de développement des patrons que celle actuelle (textuelle).

Des perspectives à court terme peuvent être de laisser le choix entre l'embarquement des composants métier correspondant à une IHM ou le téléchargement lors de l'exécution. Chacun de ces choix a ses avantages et inconvénients ; par exemple dans notre cas, l'embarquement des composants sur la plate-forme cible réduit la charge du réseau de connexion alors que les ressources mémoire sur la plate-forme sont plus occupées (la mémoire est allouée par les instances des composants). Le choix entre les deux techniques peut être réalisé en se basant sur les caractéristiques de la plate-forme cible (en particulier la capacité de la mémoire vive et la disponibilité et le débit de connexion).

Le développement des composants métier composites est un processus complexe prenant en compte la cohérence entre les connaissances du composant composite et celles des sous-composants (cf. Figure V.2) ; alors que les composants composites peuvent diminuer le temps de conception du système puisque les sous-composants sont déjà reliés.

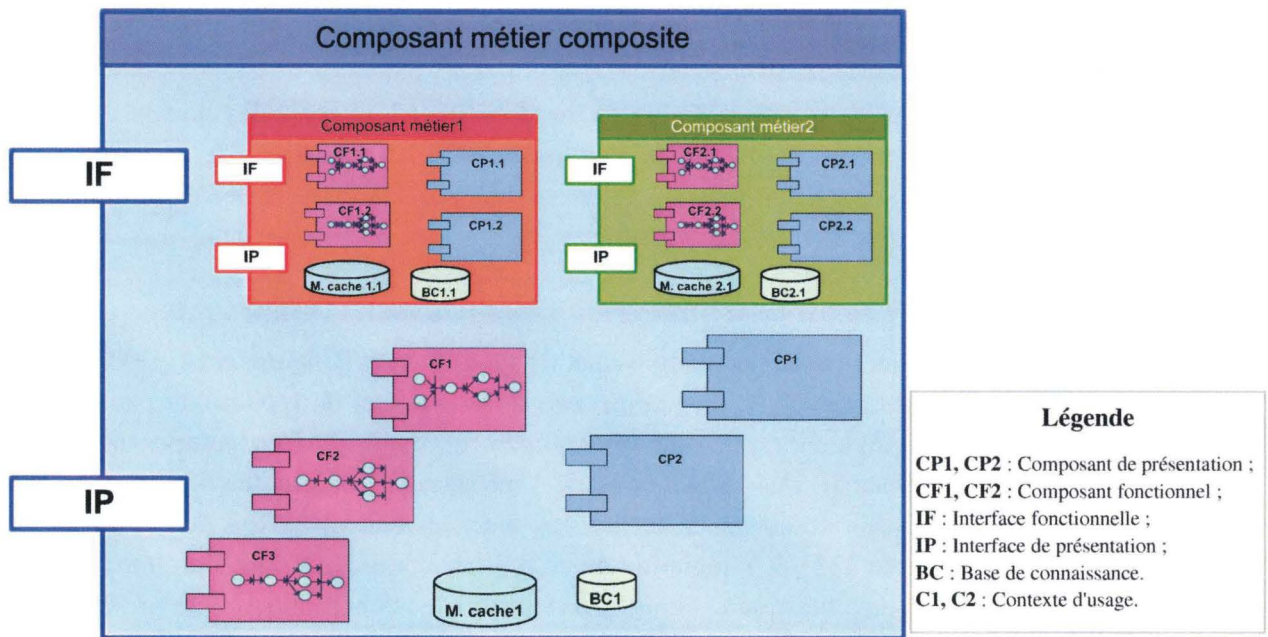


Figure V.2 : Exemple de composant métier composite

Nous avons constaté la simplicité de nos règles de choix de composants de présentation. Etendre le corpus de règles passe également par l'intégration de moteurs d'inférence plus performants, gérant le choix de composants de présentation en se basant sur une base de connaissance (composée de règles modélisant la connaissance du domaine considéré) et une autre de faits (contenant les informations concernant le cas à traiter). Les deux possibilités d'intégration du moteur d'inférence dans le système (à l'intérieur des composants ou à l'extérieur en commun entre les composants) devront être étudiées en profondeur en illustrant les avantages et les inconvénients de chacune.

V.1.3 Perspectives relatives à l'apprentissage et au capture de contexte

Des perspectives de recherche à ce travail consisteront à étudier plus en profondeur le système d'apprentissage. Le système d'apprentissage actuel nécessite chaque fois une intervention de l'expert lors de la validation des nouvelles connaissances (l'intervention peut être à distance). Par conséquent l'apprentissage peut gêner l'utilisateur. De plus le processus d'apprentissage de capture de contexte ne prend pas en compte par exemple la puissance du support cible ; par conséquent une machine avec une faible puissance de calcul, l'apprentissage et la capture de contexte peuvent ralentir l'exécution du système interactif. La solution qui peut être étudiée à moyen terme, prendrait en compte les caractéristiques de la plate-forme, en réglant l'intervalle temporel entre deux processus de capture de contexte ainsi qu'entre ceux d'apprentissage selon la puissance de la machine.

V.1.4 Perspectives relatives à la validation de la méthode

Enfin, des perspectives de recherche pourraient s'appuyer sur l'étude de cas plus complexes que ceux traités dans le chapitre IV (ex. Contrôle aérien, contrôle de centrale nucléaire...). Ceci nous permettrait de continuer à tester notre méthode.

V.2 À moyen terme

A moyen terme, des perspectives de recherche consistent à intégrer des nouveaux types des composants permettant d'adapter l'environnement d'utilisation, et à étudier en profondeur la migration d'IHM du point de vue de l'architecture de système interactif plastique proposée dans cette thèse.

V.2.1 Perspectives relatives à l'intégration des nouveaux composants

A moyen terme, l'intégration des composants de type *Wcomp* [Cheung *et al.*, 2003] dans notre méthode peut être étudiée. Les composants *Wcomp* sont de type mixte ; autrement dit un composant logiciel représente un équipement matériel. Ces composants nous permettraient d'effectuer des adaptations à l'interface physique (environnement de travail) ; par exemple : allumage de la lumière dans un bureau en cas de baisse de la luminosité (cf. Figure V.3). L'intégration de ce type de composant dans l'architecture du système informatique plastique, nous permettrait de conditionner l'environnement d'utilisation. L'adaptation de l'environnement d'utilisation s'accomplirait ou parfois remplacerait l'adaptation d'IHM. Par exemple, si un utilisateur travaille à son bureau dans une ambiance ensoleillée, et que soudain il y a un changement de luminosité provenant de l'extérieur, le système doit s'adapter à ce changement. Dans la méthode que nous avons proposée, une solution de changement des couleurs de l'IHM peut certes être appliquée ; cependant une solution complémentaire relativement simple pourrait aussi être applicable à l'environnement d'utilisation grâce aux composants *Wcomp*. Cette solution conduirait à l'allumage automatique de la lumière en ne laissant pas l'utilisateur gêné par le niveau de luminosité insuffisant.

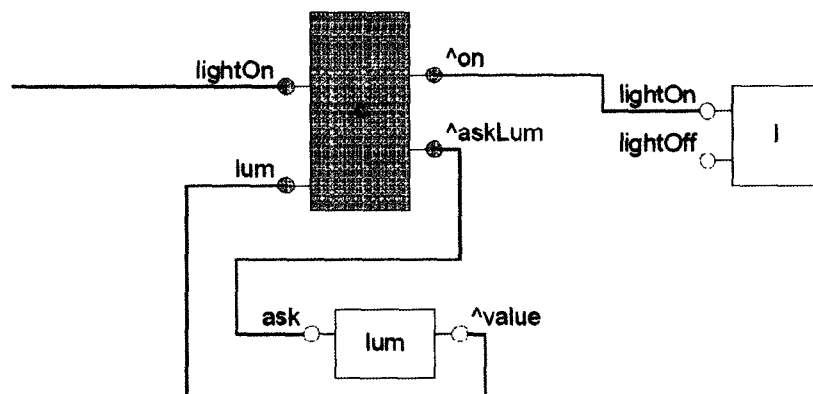


Figure V.3 : Exemple de composant *Wcomp* (Capteur et contrôle de l'allumage)
[Cheung *et al.*, 2003]

V.2.2 Perspectives relatives à la migration d'IHM

Parallèlement aux poursuites de travaux sur la notion de migration d'IHM, nous envisageons à moyen terme des travaux sur la réalisation technique de la migration. Nous avons défini au chapitre III une notation pour effectuer l'adaptation lors d'un changement contextuel, alors que la migration d'IHM suggère la nécessité de techniques permettant de

transformer le système interactif plastique vers une autre plate-forme. La Figure V.4 illustre un exemple de la manière selon laquelle la migration d'IHM peut être effectuée. Tout d'abord, le terminal actuel de l'utilisateur est affiché au milieu, et autour de lui les terminaux à proximité sont placés (la couleur de l'icône d'un terminal peut être changée selon la distance du terminal de l'utilisateur actuel). L'utilisateur peut demander d'actualiser les informations sur les terminaux, ou de migrer une IHM (en totalité ou partiellement) de son terminal actuel à un autre en projetant l'IHM concernée sur le nouveau terminal.

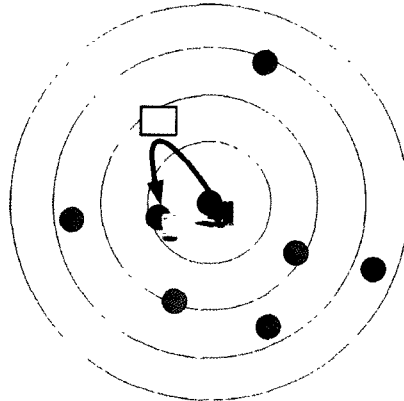


Figure V.4 : Proposition d'une manière de migrer l'IHM (côté utilisateur)

Pendant la migration et selon l'architecture proposée au chapitre III § V.6, la migration peut se dérouler à plusieurs niveaux :

- Au niveau de l'IHM, la migration peut être totale : l'intégralité de l'IHM est alors transférée sur un autre dispositif. Elle peut aussi être partielle : seulement une partie de l'IHM est transférée ; par exemple, une partie de l'IHM concernant une ou plusieurs facettes de présentation est transférée vers un PDA, mais l'IHM en totalité reste sur la plate-forme d'origine.
- Au niveau des composants métier : la structure métier peut migrer vers un autre dispositif totalement ou partiellement, en cohérence avec le précédent niveau (niveau de l'IHM). Les composants métier correspondant au système interactif plastique peuvent également rester sur la plate-forme d'origine, alors que l'IHM soit transférée toute seule vers l'autre dispositif. Lors d'une communication nécessaire entre les composants de présentation et les composants métier, un échange d'informations entre les deux dispositifs (celui d'origine et l'autre accueillant l'IHM migrante) peut être effectué.
- Au niveau du noyau fonctionnel : les composants fonctionnels peuvent être transférés partiellement ou totalement vers une autre plate-forme. Si la plate-forme d'accueil est moins puissante que celle d'origine, le noyau fonctionnel peut rester sur la plate-forme d'origine en permettant d'effectuer les calculs sur cette dernière ; de plus la migration partielle permet d'effectuer une partie de calculs sur

la plate-forme d'origine (la plus puissante). La migration partielle devrait selon nous laisser aux mécanismes de capture de contexte et d'apprentissage le choix relativement à la migration.

- Au niveau des patrons de conception : la bibliothèque des patrons de conception peut être transférée sur une autre plate-forme ou rester sur celle d'origine (afin d'éviter de surcharger la plateforme d'accueil en cas de migration d'un PC vers un téléphone portable par exemple).

Des travaux de recherche portent aujourd'hui sur le couplage entre plusieurs IHM. Ainsi dans [Barralon *et al.*, 2005], on propose une solution permettant grâce aux capteurs de proximité (détection des terminaux voisins) de réaliser le couplage de l'espace de travail. Au sein de l'espace de couplage, l'utilisateur peut, par exemple, faire migrer les IHM. [Calvary *et al.*, 2006] distinguent deux catégories d'IHM :

- L'IHM centralisée, qui s'étale sur un seul écran d'affichage ;
- L'IHM distribuée : dans ce cas l'IHM n'est plus concentrée en un unique écran d'affichage ; elle peut s'étaler sur un ensemble de plates-formes (ex. un PC avec un PDA).

Selon les deux types d'IHM ci-dessus, et dans le cadre d'une migration vers une nouvelle plate-forme, l'IHM peut passer d'un état centralisé à un autre centralisé, d'un distribué à un distribué, d'un centralisé à un autre distribué ou encore d'un état distribué à celui centralisé.

Enfin, [Bandelloni et Paternò, 2004] ont travaillé sur un principe de support de la migration de l'IHM pour les applications de Web, dans le cadre du projet Cameleon (cf. chapitre II, § III.8) ; ce principe possède une capacité de flexibilité, mais il faut s'étendre cette contribution pour prendre en compte d'autres plateformes.

V.3 À long terme : relativement à la prise en compte explicite du concept d'affordance

Notre vision à plus long terme concerne l'extension de la couverture de l'affordance et l'intégration de l'adaptation de l'environnement d'utilisation (bureau, salle de conférence, etc.).

L'affordance a été introduite par [Gibson, 1977 ; 1979] dans le domaine de la perception. La notion primitive d'affordance est la "lecture" d'une opportunité d'action dans les propriétés visuelles d'un objet (théorie des affordances visuelles). Ensuite l'affordance a été utilisée en particulier dans le domaine de la conception dite écologique des systèmes, en interprétant de fait l'affordance comme une cohérence entre les représentations mentales du sujet et le système de présentation d'information et de commandes que propose le dispositif technique [Rasmussen & Vicente, 1989 ; Vicente, 1999]. Puis, Gibson a travaillé sur l'analyse des informations à l'existence d'affordances [Gibson,

2000], analyse qui inclut « des relations invariantes entre événements qui évoluent dans le temps ». La popularisation de l'affordance a donné lieu à de nombreuses définitions du concept d'affordance en lien avec la théorie complexe de Gibson. Parmi ces définitions, celle de Norman est brève et claire : « le terme affordance fait référence aux propriétés réelles perçues d'un objet, à savoir, essentiellement, les propriétés fondamentales qui déterminent les possibilités d'utilisation de l'objet. »²⁶ [Norman, 1988].

Nous pouvons définir le concept d'affordance comme la perception d'une utilité ; ou encore le potentiel pour l'action que recèle un objet, c'est-à-dire la capacité de ce même objet à servir la volonté d'agir d'un sujet. [Norman, 1994] a opéré une distinction entre "affordance réelle" et "affordance perçue". Il a identifié deux types d'objets :

- Ceux informationnels ayant une fonction de représentation de l'action (ils visent à faciliter la manipulation et l'exécution), cette signification est liée à l'expérience perceptuelle.
- D'autres servent de supports informationnels pour l'action (ils facilitent la mémoire et le traitement des symboles).

Le concept d'affordance a été développé et est largement utilisé pour décrire les caractéristiques d'un environnement à fournir une signification pour l'usage.

Pour notre part, nous avons procédé à une première étude de la perspective d'intégration de la notion d'affordance dans la plasticité dans le cadre d'une journée de travail du GDR E HAMASYT (GDR Européen « HumAn-MAchine Systems in Transportation ») [Hariri *et al.*, 2007]. Notons que cette intégration nécessitera de préciser à terme les liens entre l'utilisabilité et l'affordance, ce qui n'est pas encore clair dans la littérature actuellement.

La perspective d'intégration de la notion d'affordance dans la méthode proposée au troisième chapitre conduit dans un premier temps à la question globale suivante : comment, quand, et où les notions d'affordance peuvent être intégrées dans notre méthode, afin de pouvoir les supporter (cf. Figure V.5) ?

²⁶ « The term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could be used. »

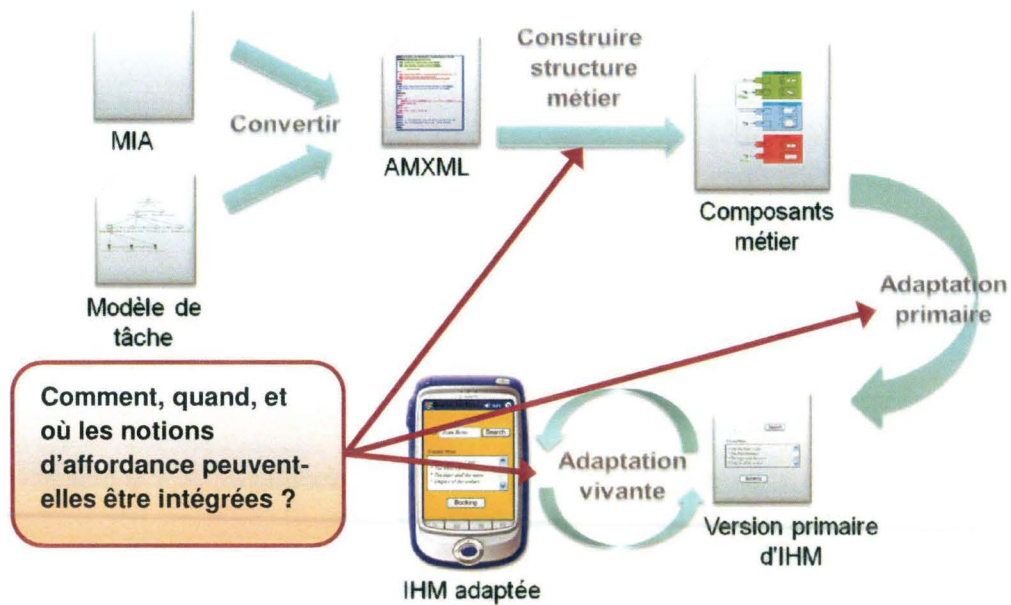


Figure V.5 : Méthode globale de génération d'IHM plastique

Au niveau de la conception, un point à étudier concerne le modèle AMXML. Il faudrait faire en sorte qu'il puisse contenir des spécifications concernant les notions d'affordances, à faire cohabiter avec celles liées à la plasticité (cf. Figure V.6). Ainsi de nouvelles propositions doivent être ajoutées à la DTD dédiée à l'étape "convertir le MIA en XML".

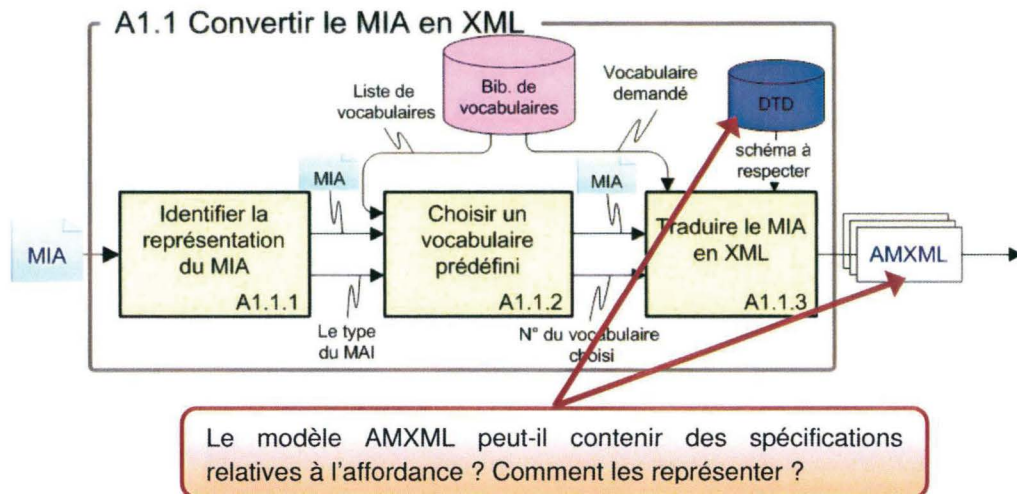


Figure V.6 : Convertir le MIA en AMXML

Pendant la construction de la structure métier du système, une étude devrait concerner les logiques de composition des composants métier. Il faudrait pouvoir les évaluer sous l'angle de l'affordances. Ensuite, il faudrait être capable de prédire leur(s) capacité(s) d'affordances (cf. Figure V.7). La Figure V.8 illustre deux compositions de composants métier pour les mêmes fonctionnalités : il s'agirait de faire correspondre à ces deux compositions deux perceptions possibles d'affordances.

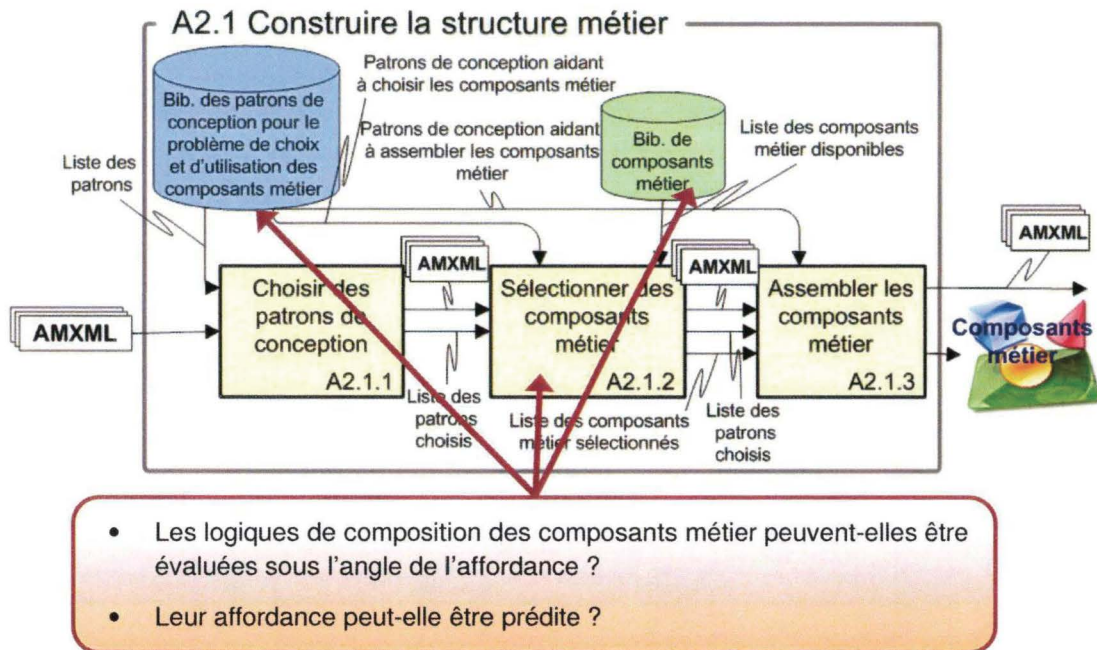


Figure V.7 : Construction de la structure métier

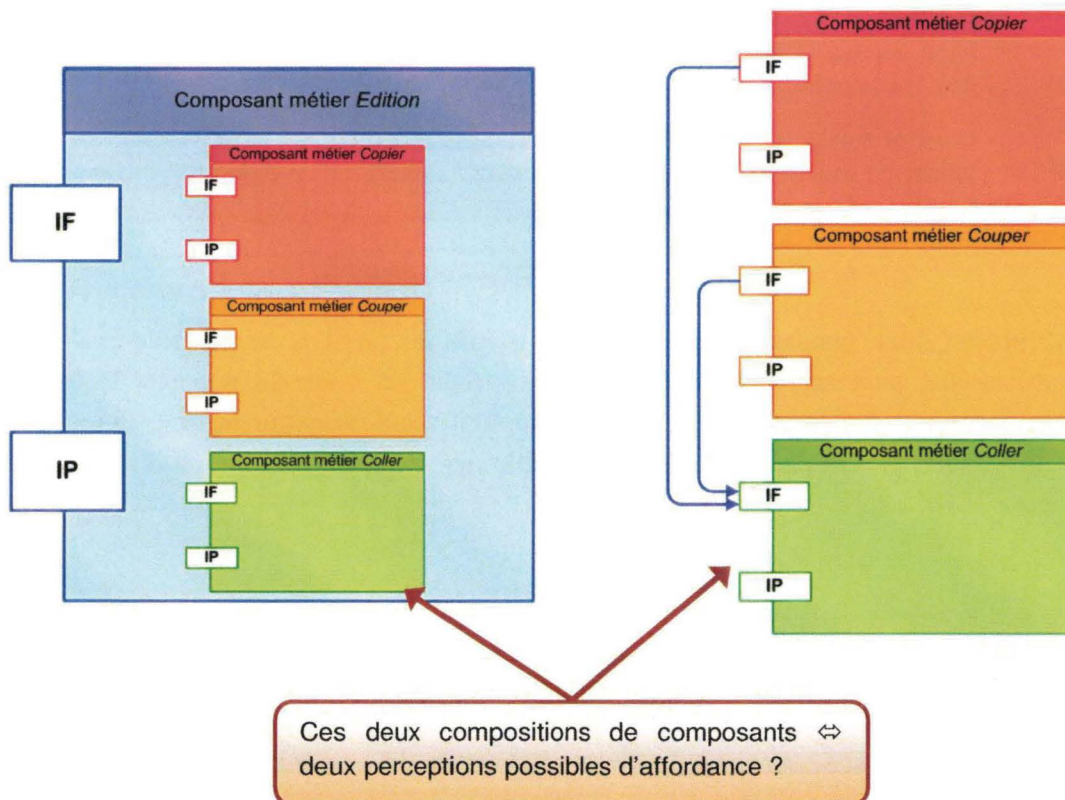


Figure V.8 : Deux compositions métier possibles

A l'adaptation primaire, d'autres points devraient être étudiés. Ainsi, il faudrait savoir comment caractériser les conditions d'affordance de chaque composant de présentation. Il faudrait aussi trouver des critères de sélection des composants de présentation, afin d'assurer un niveau d'affordance (cf. Figure V.9). Une étude supplémentaire concernerait la manière dont les patrons de conception pourraient prendre en compte l'affordance pendant l'assemblage et la configuration des composants de présentation.

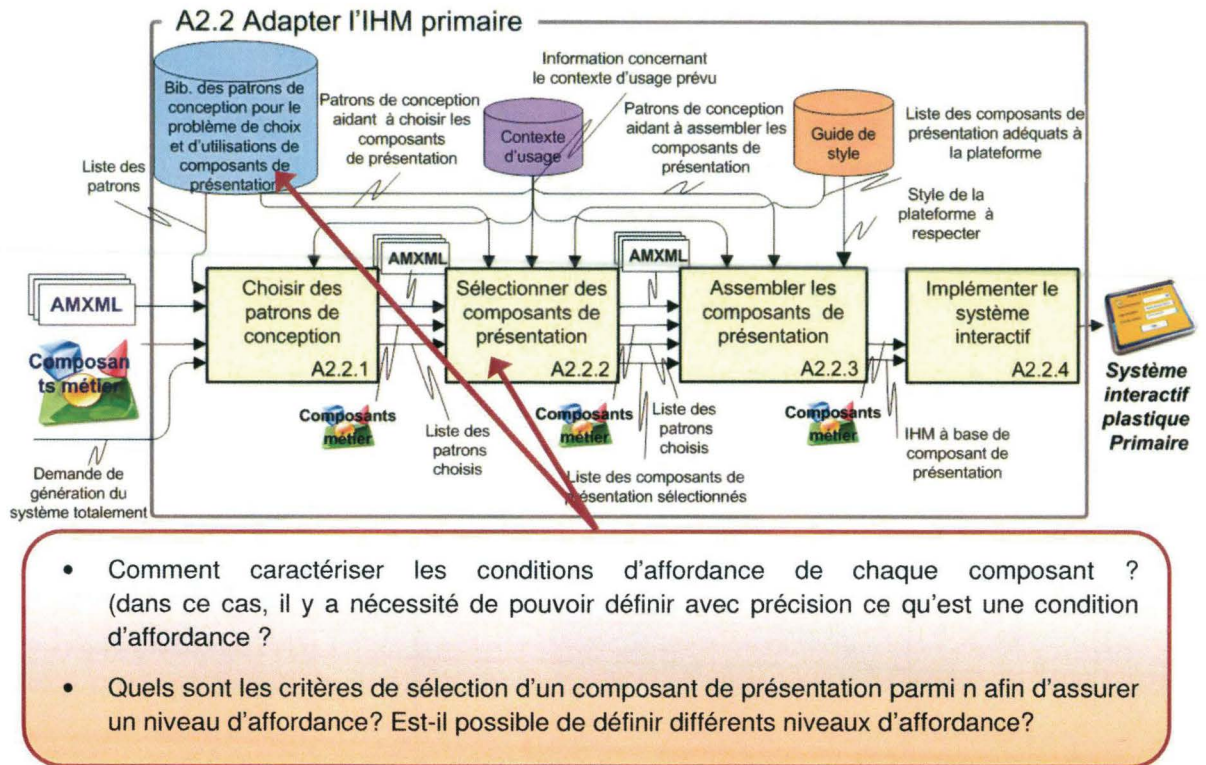


Figure V.9 : Adaptation primaire

Au niveau final, pendant l'utilisation du système interactif plastique, une évaluation de l'utilisabilité peut se lancer lors d'un changement du contexte d'usage. Dans ce cas, l'évaluation de l'utilisabilité doit inclure (implicitement ou explicitement) une évaluation d'affordance. Il faut donc définir au préalable des critères d'évaluation d'affordance (cf. Figure V.10).

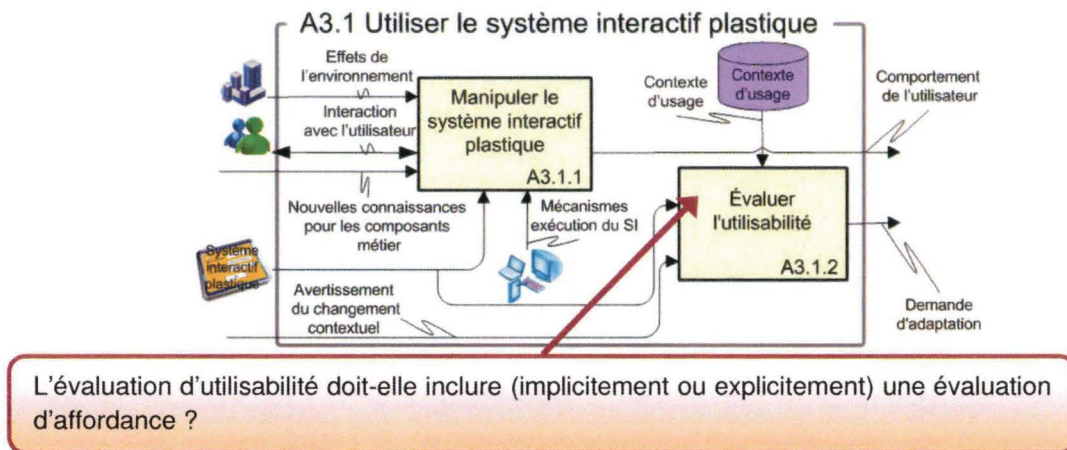


Figure V.10 : Utilisation du système interactif plastique

Lors de l'activité de capture de contexte d'usage, il est important que l'évaluation du changement contextuel puisse détecter que les changements liés au contexte auront des conséquences (positives ou négatives) sur l'affordance (cf. Figure V.11). Des mécanismes doivent donc être étudiés et développés à ce sujet.

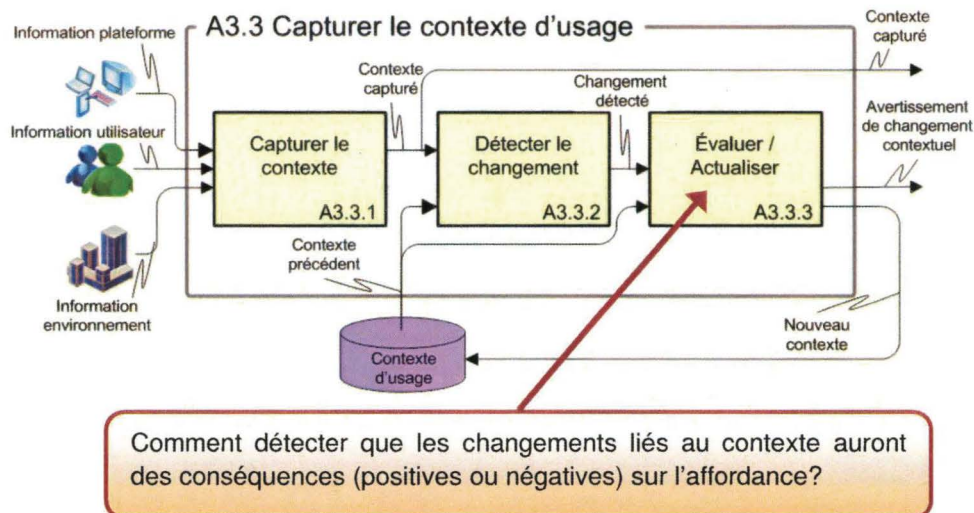


Figure V.11 : Capture du contexte d'usage

Relativement à l'apprentissage au cours de l'utilisation, plusieurs aspects doivent être étudiés. Il est d'abord important de savoir ce qu'on peut apprendre sur l'affordance (efficacité, impact...) au fur et à mesure des interactions homme-machine. Il faut aussi être capable de générer automatiquement des connaissances relatives à l'affordance, permettant ensuite et dans la mesure du possible d'en améliorer l'effet auprès des utilisateurs (voir Figure V.12).

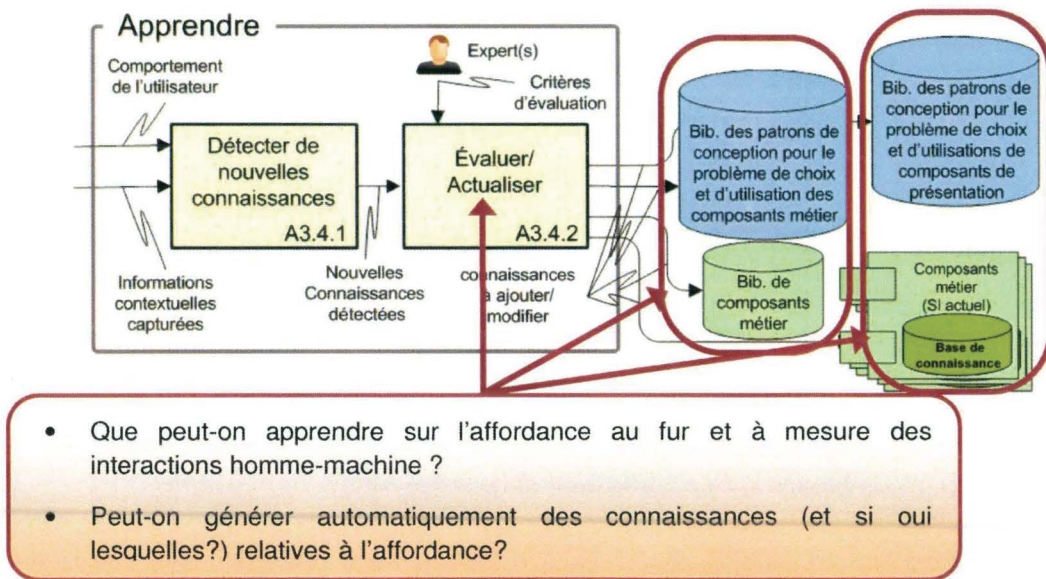


Figure V.12 : Processus d'apprentissage

Enfin, en ce qui concerne l'adaptation vivante, en cas de passage à une nouvelle plateforme, plusieurs études pourraient être menées (cf. Figure V.13). On peut ainsi s'interroger sur les conditions de dégradation de l'affordance à l'exécution. On doit aussi étudier comment considérer les changements liés au contexte pour préserver (ou même améliorer) l'affordance. Dans cette perspective, une nouvelle génération des règles d'adaptation serait nécessaire afin de préserver l'utilisabilité et l'affordance de l'IHM.

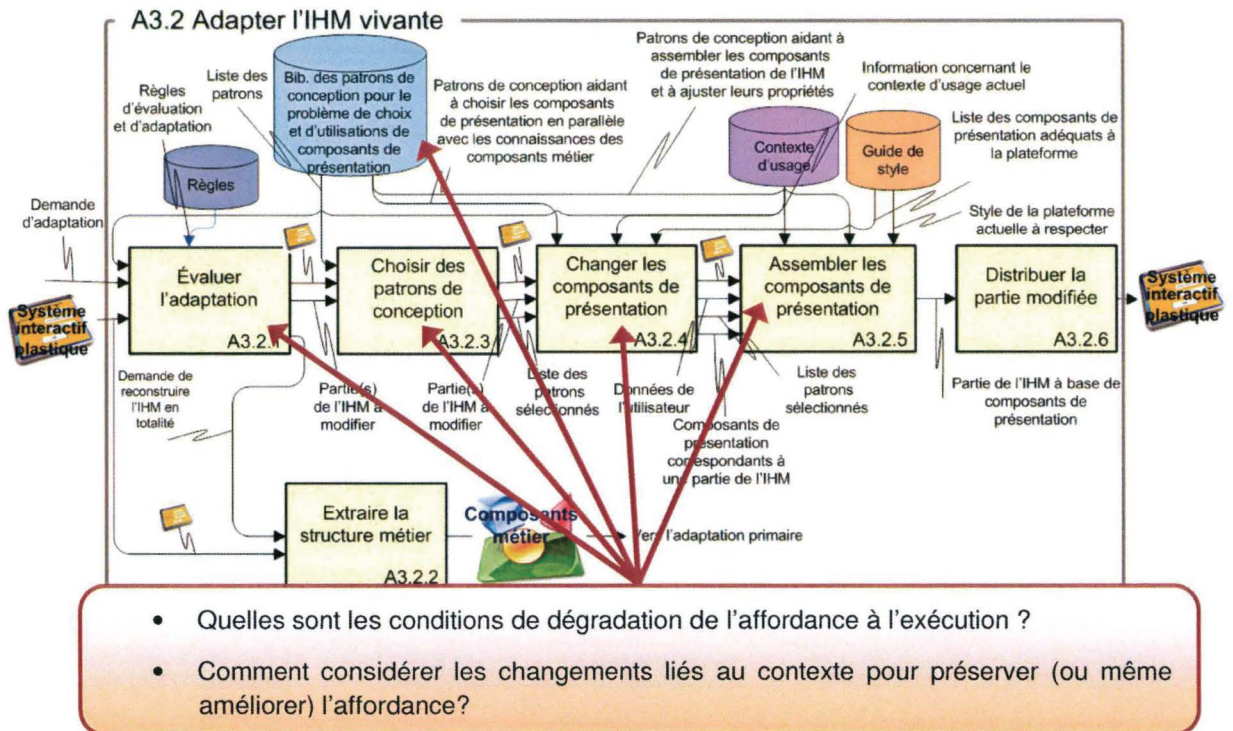


Figure V.13 : Processus d'adaptation vivante

VI. Conclusion

Dans ce chapitre, nous avons proposé une discussion sur l'originalité de notre contribution. Nous avons mis en évidence les points forts et faibles de la méthode proposée, par rapport à la conception et à l'implémentation d'IHM plastique. Cette discussion s'est basée sur les résultats de l'implémentation de la méthode sur les deux études de cas, et en nous positionnant par rapport à d'autres approches contribuant à l'état de l'art de cette thèse.

Nous pensons que différents points peuvent être améliorés, en ce qui concerne la méthode, relativement aux différents principes et mécanismes constitutifs de celle-ci. Ainsi différentes perspectives ont été proposées à ce sujet à court, moyen et long termes. Pour ce qui est du long terme, il nous semble qu'un ensemble d'études et recherches de fond pourraient être menées afin de préciser le concept d'affordance et le mettre en œuvre explicitement dans le cadre d'IHM plastiques.

Conclusion générale

Au bilan, le travail présenté dans ce mémoire contribue à la conception et la génération d'IHM plastiques à base de composants métier et à l'aide de patrons de conception. Cette contribution comporte deux principaux aspects : l'analyse bibliographique et critique de l'existant, et la proposition d'une méthode dédiée à la conception et la génération d'IHM plastiques à base de composants métier, et en s'appuyant sur la notion de patrons de conception spécifiques au domaine de l'adaptation.

Nos travaux de recherche doctorale ont contribué à spécifier et générer l'IHM plastique à partir d'un modèle abstrait et/ou d'un modèle de tâche. L'architecture du système s'appuie sur une composition basée sur les composants métier. Ceux-ci ont la capacité de changer dynamiquement leur facette de présentation. Cela est adopté comme une des solutions de l'adaptation dynamique au contexte d'usage. La méthode se base sur l'utilisation de patrons permettant de faire des choix concernant les composants métier relatifs aux tâches du système et de configurer, d'assembler et de relier les composants de présentation de manière à ce qu'ils soient adaptés au contexte d'usage. Comme les principes d'adaptation sont basés sur des décisions déjà définies dans des bases de connaissance, nous avons intégré la notion d'apprentissage dans la méthode, afin d'améliorer la qualité des décisions à différents niveaux. La méthode a été appliquée sur deux exemples concrets : la première étude de cas concerne un système applicable dans la vie quotidienne, et une seconde étude provient du monde industriel. Les études de cas montrent une connexion possible entre les méthodes classiques de spécification et conception des systèmes interactifs et la notion de plasticité. De plus elles illustrent la nécessité de l'adaptation vivante avec la prise en compte des nouvelles connaissances afin de préserver l'utilité de l'adaptation. Ces deux aspects sont relativement absents dans les méthodes et langages étudiés dans l'état de l'art (chapitres I et II).

Ainsi, à travers le premier chapitre, nous avons présenté sans souci d'exhaustivité mais plutôt de représentativité, des architectures de systèmes interactifs. Cette étude a d'abord décrit les architectures recensées dans la littérature. Les architectures centralisées décomposent l'interface en différents niveaux logiciels visant une séparation nette entre l'interface et l'application. D'autres approches qualifiées de réparties, appelées aussi orientées agent, ont été abordées. Le troisième type de modèle d'architecture qualifié d'hybride tente de tirer profit des deux types d'architectures précédentes en reprenant les principaux avantages de chacun d'eux. Dans un deuxième temps, nous avons abordé les patrons de conception, qui constituent pour nous une base de solution pour la conception d'IHM plastique. Nous avons décrit des approches orientées composants en montrant les divers types de composants et de compositions architecturales. Nous avons mis en évidence (1) la composition de type statique qui ne peut être modifiée, dans un but d'adaptation au nouveau contexte, que par le concepteur lors de la phase de conception, et (2) la composition dynamique, celle-ci étant supportée par des mécanismes permettant de remplacer ou de changer des unités algorithmiques ou structurelles à l'exécution. Nous avons distingué deux types d'approches à composition dynamique dont le noyau

fonctionnel du système peut être modifié ou reconfiguré selon différentes capacités de dynamicité. Le premier chapitre s'est terminé par une description générale de l'informatique pervasive avec ses différentes caractéristiques du point de vue de l'IHM, pour aboutir à la présentation de la plasticité de l'IHM. Il a été montré que "la plasticité était devenue une nécessité" [Calvary et Coutaz, 2002a] face aux nouveaux environnements d'utilisation non homogènes, à la variété des profils des utilisateurs, et aux diverses situations possibles à considérer.

Dans le deuxième chapitre, nous avons exposé les principales méthodes et langages proposés dans la littérature pour la spécification et la génération d'IHM multiplateforme ou orientée contexte, en mettant en évidence leurs apports et leurs limites. Nous avons vu que les cas élémentaires de plasticité sont ceux où l'adaptation ne prend en compte que le contexte lié à la plate-forme. En dehors de ces cas élémentaires, des travaux de recherche se sont focalisés sur l'adaptation pré-calculée supportée par le retour à la conception lors d'un important changement contextuel. A notre connaissance, les méthodes présentées ne possèdent aucun mécanisme clairement défini d'amélioration de la qualité de réponse aux changements contextuels. Un certain nombre de critères ont permis de comparer les méthodes et langages.

Ainsi, nous avons proposé au troisième chapitre une méthode de conception et génération d'IHM plastique. L'IHM plastique générée possède une capacité d'adaptation dynamique applicable à l'exécution, et prend en considération la plus grande gamme possible de chaque élément du contexte d'usage. La méthode proposée s'appuie au départ sur un modèle d'IHM abstraite et/ou un modèle de tâche. Les connaissances de conception sont potentiellement représentées sous forme de patrons. Les patrons de conception sont utilisés dans les différentes étapes de la méthode afin d'effectuer le choix et l'assemblage des composants métier, et la décoration des composants de présentation. Ainsi, ils peuvent être appelés à l'exécution afin de supporter l'adaptation de l'IHM. Une architecture dédiée aux systèmes interactifs plastiques a été proposée, celle-ci s'appuyant sur une composition basée sur les composants métier. Un nouveau type de composant métier, dédié à l'adaptation, a été proposé ; il a la capacité de changer dynamiquement sa facette de présentation à l'exécution. Le choix de la présentation s'effectue à l'aide d'une base de connaissance intégrée dans l'architecture du composant métier. L'intégration d'une technique d'apprentissage permet de continuer à développer la base de connaissance du système à l'exécution afin de préserver l'utilité de l'adaptation.

Le quatrième chapitre a permis de présenter une mise en application de la démarche sur deux cas d'étude destinés à deux environnements d'utilisation différents. Premièrement, une IHM d'un système de guidage touristique a été spécifiée et générée en partant d'un modèle de tâche représenté en CTT. Le modèle AMXML a été généré en appliquant les règles de construction d'AMXML (cf. chapitre III). Le deuxième cas d'étude prend la forme d'un système de supervision d'un processus chimique. Un modèle d'IHM de TOOD a été représenté pour construire le modèle AMXML, en utilisant un vocabulaire spécifique. Des processus d'adaptation vivante et aussi d'apprentissage ont été lancés lors de l'utilisation des IHM générées par notre méthode. Les contextes d'usage des deux cas

d'études ont été expliqués avec des illustrations. L'implémentation de notre méthode sur les deux cas d'étude a montré la nécessité de l'adaptation vivante et de la prise en compte des nouvelles connaissances. Autrement, l'utilité de l'adaptation vivante aurait été perdue progressivement. De plus, grâce à un vocabulaire spécifique, notre méthode a réussi à réutiliser des spécifications abstraites d'IHM dans une méthode classique à générer des IHM plastiques.

L'illustration de la méthode sur ces deux cas d'étude différents nous a permis de distinguer des points forts et faibles de la méthode, décrits dans le cinquième chapitre. En effet, une critique et des perspectives de recherches ont été fournis. En ce sens, nos perspectives à court terme sont plutôt d'ordre implémentatif, en continuant à développer une infrastructure logicielle supportant la méthode proposée avec la possibilité d'intégration des outils existants d'autres méthodes (ex : outils d'UsiXML), et en implémentant notre méthode sur des exemples plus complexes. Celles à moyen terme correspondent à l'intégration d'autres catégories de composants permettant, par exemple, d'adapter l'environnement d'utilisation (interface physique), et de travailler en profondeur sur la notion de migration d'IHM (migration vers une autre modalité ou un autre canal d'interaction). Enfin, nos perspectives à long terme s'intéressent à la prise en compte explicite du concept d'affordance dans la conception et la génération d'IHM plastiques.

Références bibliographiques

- Abowd G., Atkeson C.G., Hong J., Long S., Kooper R. and Pinkerton M. (1996). A Mobile Context-Aware Tour Guide. GVU Technical Report GIT-GVU-96-27, décembre.
- Abrams M., Phanouriou C., Batongbacal A. L., Williams S. M. and Shuster J.E. (1999). UIML: An Appliance-Independent XML User Interface Language. In *Computer Networks*, Vol. 31, pp. 1695–1708.
- Abowd G. and Mynatt E.D. (2000). Charting Past, Present, and Future Research in Ubiquitous Computing, *ACM Trans. Computer-Human Interaction*, 7(1), pp. 29–58.
- Abrams M. and Helms J. (2004). User Interface Markup Language (UIML) Specification version 3.1, Technical report, Harmonia.
- AFNOR (2006). Symboles graphiques et pictogrammes - Couleurs de sécurité et signaux visuels de sécurité - Partie 1 : principes de conception. Norme NF X08-003-1 Juillet 2006, Association Française de Normalisation.
- Alexander C., Ishikawa S., Silverstein M., Fiskdahl-King I. and Angel S. (1977). A pattern language. Oxford University Press, New York.
- Ali M.F., Pérez-Quñones M.A. and Abrams M. (2004). Building Multi-Platform User Interfaces with UIML. In Seffah A. and Javahery H. (Eds.), *Multiple User Interfaces: Engineering and Application Framework*. John Wiley, Chichester, pp. 95–118.
- Allen R. and Garlan D. (1997). A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering*, 6(3), pp. 213–249.
- Allen P. and Frost S. (1998). *Component-based Development for Enterprise Systems, Applying the Select Perspective*. Cambridge University Press.
- Ang C.L. (1999). Enactment of IDEF0 Modes. *International Journal of Production Research*, 37(15), pp. 3383–3397.
- Bandelloni R. and Paternò F. (2004). Migratory user interfaces able to adapt to various interaction platforms. *Int. J. Hum.-Comput. Stud.*, 60(5–6), pp. 621–639.
- Barais, O. and Duchien, L. (2004). TranSAT : Maîtriser l'Evolution d'une Architecture Logicielle. *L'Objet*, 10(2–3), pp. 103–116.
- Barbier F., Cauvet C., Oussalah M., Rieu D., Bennisri S. and Souveyet C. (2002). Composants dans l'ingénierie des systèmes d'information : concepts clés et techniques de réutilisation. *Information Interaction Intelligence, Actes des 2e Assises nationales du GDRI3*, Nancy, Cepaduès, pp. 95–117.
- Bass L., Mittle R., Pellegrino R., Reed S., Seacord S., Sheppard S. and Szesur M. (1991). The Arch Model : Seeheim revisited. In *Proceedings of User Interface Developer's Workshop*, Seeheim.
- Beach B.W. (1992). Connecting Software Components with Declarative Glue. In *Proc. of ICSE'92*, Melbourne, May 1992. ACM Press. pp. 120–137.
- Blair G.S., Coulson G., Robin P. and Papathomas M. (1998). An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, The Lake District, England, septembre.

- Blay-Fornarino M., Ensellem D., Ocelllo A., Pinna-Dery A.M., Riveill M., Fierstone J., Nano O. and Chabert G. (2002). Un service d'interactions : principes et implémentation. In Journée composants : Systèmes à composants adaptables et extensibles. Grenoble, France, octobre. (Prototype accessible à l'adresse : <http://rainbow.essi.fr/noah>)
- Blay-Fornarino M., Emsellem D., Pinna-Dery A.M. and Riveill M. (2004). Un service d'interactions : principes et implémentation. *Technique et science informatiques RSTI série TSI (RSTI-TSI)*, 23 (2), pp. 175–204.
- Booch G., Rumbaugh J. and Jacobson I. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley.
- Borchers J. (2001). *A pattern approach to interaction design*. John Wiley & Sons Ltd, England.
- Bouillon L., Vanderdonckt J. and Chieu Chow K. (2004). Flexible Re-engineering of Web Sites. In proceedings of the international conference on Intelligent User Interfaces (IUI'04), January 13–16, Madeira (Portugal), ACM Press, New York, USA, pp. 132–139.
- Bradbury J.S., Cordy J.R., Dingel J. and Wermelinger M. (2004). Supporting Self-Management in Dynamic Software Architecture Specifications. Proc. WOSS'04, Workshop on Self-Managed Systems, at SIGSOFT 2004 / FSE-12 - ACM SIGSOFT 12th International Symposium on Foundations of Software Engineering, Newport Beach, California, pp. 28–33.
- Bradshaw J.M. (1997). *Software agent*. MIT Press, Cambridge, USA.
- Brown P., Bovey J. and Chen X. (1997). Context-aware applications: from the laboratory to the market place. *IEEE Personal Communications*, 2(1), pp. 1–9.
- Brown W.J., Malveau R.C., Brown W.H., Iii H.W.M. and Mowbray T.J. (1998). *Anti Patterns : Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1st édition.
- Budinsky F.J., Finnie M.A., Vlissides J.M. et Yu P.S. (1996). Automatic Code Generation from Design Patterns. *IBM Systems Journal*, 35(2), pp.151–171.
- Calvary G., Coutaz J. and Thevenin D. (2001a). A Unifying Reference Framework for the Development of Plastic User Interfaces. IFIP WG 2.7/13.2, EHCI01, Toronto, Canada.
- Calvary G., Coutaz J. and Thevenin D. (2001b). Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism. In Blandford A., Vanderdonckt J. and Gray P. (Eds.), Springer-Verlag, London, pp. 349–363.
- Calvary G. and Coutaz J. (2002a). *Plasticité des Interfaces : Une Nécessité !*. Acte des assises nationales du GDR I3, Nancy, décembre.
- Calvary G., Coutaz J., Thevenin D., Bouillon L., Florins M., Limbourg Q., Souchon N., Vanderdonckt J., Marucci L., Paternò F. and Santoro C. (2002b). *The CAMELEON Reference Framework*. Deliverable D1.1.
- Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L. and Vanderdonckt J. (2003). A unifying reference framework for multi-target user interface. *Interacting with Computers*, 15 (3), pp. 289–308.

- Calvary G., Coutaz J., Dâassi O., Balme L. and Demeure A. (2004). Towards a New Generation of Widgets for Supporting Software Plasticity: the “Comet”. In Proceedings of EHCI-DSVIS'04, The 9th IFIP Working Conference on Engineering for Human-Computer Interaction, Hamburg, Germany, juillet.
- Calvary G., Dâassi O., Coutaz J. and Demeure A. (2005). Des Widgets aux Comets pour la Plasticité des Systèmes Interactifs. *Revue d'Interaction Homme-Machine (RIHM)*, 6(1), pp. 33–53.
- Calvary G., Coutaz J., Daassi O., Ganneau V., Balme L., Demeure A. and Sottet J.S. (2006). Métamorphose des IHM et Plasticité : Article de synthèse. Actes de Ergo'IA 2006, Bidart/Biarritz, France, octobre.
- Cheng S.W., Huang A.C., Garlan D., Schmerl B. and Steenkiste P. (2004). Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure. *Computer*, 37(10), pp. 46–54.
- Cheung D., Fuchet J., Grillon F., Joulie G. and Tigli J.Y. (2003). Wcomp : Rapid Application Development Toolkit for Wearable computer based on Java. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (Hyatt Regency, Washington, D.C., USA, Oct 2003), volume 5, pp. 4198–4203.
- Chung L. (1991). Representation and Utilization of Non-functional Requirements for Information System Design. In Proceedings of the 3rd international conference on Advanced Information System Engineering – CaiSE'91, Trondheim, Norway, mai.
- Clark J. (1999). XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium Recommendation, novembre. (<http://www.w3.org/TR/xslt>)
- Clerckx T., Vandervelpen C., Luyten K. and Coninx K. (2007). A Prototype-Driven Development Process for Context-Aware User Interfaces. In the proc. of 5th International Workshop on TAsk MODels and DIAGrams for user interface design TAMODIA'2006 (Hasselt, Belgium, October 23-24, 2006), Coninx K., Luyten K. and Schneider K.A. (Eds.), Lecture Notes in Computer Science (LNCS) 4385, Springer-Verlag, Berlin, pp. 339-354, 2007.
- Conte A., Giraudin J.P., Hassine I. and Rieu D. (2001). Un environnement et un formalisme pour la définition, la gestion et l'application de patrons. *Revue Ingénierie des Systèmes d'Information*, numéro spécial, 6(2), Hermès.
- Conte A., Fredj M., Hassine I., Giraudin J.P. and Rieu D. (2002a). AGAP : an environment to design and apply patterns. IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC), Hammamet, Tunisia, octobre.
- Conte A., Fredj M., Hassine I., Giraudin J.P. and Rieu D. (2002b). A tool and a formalism to design and apply patterns. 8th International Conference On Information Systems OOIS'02, Montpellier, France.
- Cornuéjols A., Miclet L. and Kodratoff Y. (2002). Apprentissage artificiel. Concepts et algorithmes, Eyrolles.
- Coutaz J. (1987). PAC: an Implementation Model for Dialog Design. In Proceedings of Interaction, Bullinger H.J. and Shackel B. (Eds.), North Holland, Stuttgart, pp. 431–436.
- Coutaz J. (1990). Interface homme-ordinateur : conception et réalisation. Dunod Informatique Ed., Paris.

- Coutaz J. and Nigay L. (2001). Architecture logicielle conceptuelle des systèmes interactifs. In Kolski C. (Ed.), *Analyse et conception de l'IHM*, Hermès Science Publications, Paris, pp. 207–246.
- Coutaz J., Crowley J., Dobson S. and Garlan D. (2006). Context is Key. *Communication of the ACM*, 48(3), pp. 49–53.
- Crease M. (2001). *A Toolkit of Resource-Sensitive, Multimodal Widgets*. Thèse de doctorat, Department of Computing Science, University of Glasgow, Decembre.
- D'Souza D. (2003). *Component and Catalysis Cost-Effective Software Development*. (www.catalysis.org).
- Dâassi O., Calvary G., Coutaz J. and Demeure A. (2003). Comet : Une nouvelle génération de "widget" pour la Plasticité des Interfaces. In *proc. of conférence IHM 2003* (Caen, France, novembre 2003), Baudel T. (Ed.), Vol. 51, ACM press, New York, pp. 64–71.
- Demeure A. (2007). *Modèles et outils pour la conception et l'exécution d'Interfaces Homme-Machine Plastiques*. Thèse de doctorat, Université Joseph Fourier - Grenoble I, octobre.
- Depaulis F. (2002). *Vers un environnement générique d'aide au développement d'applications interactives de simulations de métamorphoses*. Thèse de doctorat, Université de Poitiers, novembre.
- Dey A.K. (2000). *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology.
- Dey. A.K. (2001). Understanding and Using Context. Special issue on Situated Interaction and Ubiquitous Computing, *Personal Ubiquitous Comput.*, 5(1), pp. 4–7.
- Dubinko M., Leigh L., Klotz J., Merrick R. and Raman T.V. (2003). XForms1.0., W3C Recommendation, World Wide Web Consortium, Octobre 2003. (<http://www.w3.org/TR/xforms/>)
- Duke D.J. and Harrison M.D. (1993). Abstract Interaction Objects. *Computer Graphics Forum*. NCC Blackwell, 12(3), pp.25–36.
- Durkin J. (1994). *Expert System design and development*. Prentice Hall.
- Eisentien J. and Puerta A. (1998). TIMM: Exploring Task-Interface Links in MOBI-D. In *CHI'98 Workshop on From Task to Dialogue: Task-Based User Interface Design*, Los Angeles, avril.
- Eisenstein, J. and Puerta, A. (2000). Adaptation in automated user-interface design. In *Proceedings of the 2000 Conference on Intelligent User Interfaces*, New Orleans, 9–12 January 2000, ACM Press, pp. 74–81.
- Eisenstein J., Vanderdonckt J. and Puerta A. (2001). Model-Based User-Interface Development Techniques for Mobile Computing. In *Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001* (Santa Fe, January 2001), Lester J. (Ed.), ACM Press, New York, pp. 69–76.
- Experience Dynamics Corp. (2007). *Science of Usability -User Interface Style Guides*. http://www.experiencedynamics.com/science_of_usability/ui_style_guides/ (Mis à jour en août 2007).

- Faconti G. and Paternò F. (1990). An approach to the formal specification of the components of an interaction. In Vandoni C. and Duce D. (Eds.), Eurographics 90, pp. 481–494, North-Holland.
- Fekete J.D. (1996). Un modèle multicouche pour la construction d'applications graphiques interactives. Thèse de doctorat, Université de Paris XI, janvier.
- Ferber J. (1995). Les systemes multi-agents. InterEditions.
- Findlater L. and McGrenere J. (2008). Impact of screen size on performance, awareness, and user satisfaction with adaptive graphical user interfaces. In Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI 2008), ACM Press, pp. 1247-1256.
- Fitzpatrick T., Blair G., Coulson G., Davies N. and Robin P. (1998). A software architecture for adaptive distributed multimedia applications. IEE Proceedings Software, 145(5), pp. 163–171.
- Florins M., Simarro F.M., Vanderdonck J, and Michotte B. (2006). Splitting rules for graceful degradation of user interfaces. In Proc. of the Working Conf. on Advanced Visual interfaces AVI '06, (Venezia, Italy, May 23-26, 2006), ACM Conference Proceedings Series, pp. 59-66.
- Foley J.D. and Van Dam A. (1982). Fundamentals of Interactive Computer Graphics. Addison-Wesley Ed.
- Foley J.D., Wallace V.L. and Chan P. (1984). The Human Factors of Computer Graphics Interaction Techniques. IEEE computer Graphics and Applications, 4(11), pp.13–48.
- Forbrig P., Dittmar A., Reichart D. and Sinnig D. (2004). From Models to Interactive Systems Tool Support and XI ML. Proceedings of First International Workshop on Making model-based user interface design practical: usable and open methods and tools, MBUI'04, Funchal, Madeira, Portugal, janvier.
- Frost S. (1995). The select perspective V. 4.0, Technical report.
- Futtersack M. and Labat J. M. (2000). Quelle planification pédagogique dans les EIAH ?. Revue STE, N° spécial « Education et Informatique, hommage à Martial Vivet » Ed., vol 7, Hermès, pp. 165–177.
- Gachet A. and Haettenschwiler P. (2003). A jini-based software framework for developing distributed cooperative decision support systems. Software-Practice and Experience, 33(3), pp. 221–258.
- Gaffar A., Sinnig D., Seffah A. and Forbrig P. (2004). Modeling patterns for task models. In TAMODIA'04: Proceedings of the 3rd annual conference on Task models and diagrams, New York, USA, ACM Press, pp. 99–104.
- Gajos K.Z., Wobbrock J.O. and Weld D.S. (2007). Automatically generating user interfaces adapted to users' motor and vision capabilities. In Proc. of the 20th Annual ACM Symposium on User interface Software and Technology UIST '07 (Newport, USA, October 07-10, 2007), ACM Press, New York, pp. 231-240.
- Gamma E., Helm R., Johnson R. and Vlissides J. (1994). Design Patterns, Catalogue de modèles de conception réutilisables. Vuivert, Paris.
- Gamma E., Helm R., Johnson R. Vlissides J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

- Garlan D., Schmerl B. and Chang J. (2001). Using gauges for architecture-based monitoring and adaptation. In Proc. of the Working Conference on Complex and Dynamic Systems Architecture, Brisbane, Australie, december.
- Geier M., Steckermeier M., Becker U., Hauck F.J., Meier E. and Rasthofer U. (1998). Support for mobility and replication in the AspectIX architecture. Tech. Rep. TR-I4-98-05, University of Erlangen-Nuernberg, IMMD IV.
- Gibson E.J. (2000). Where is the information for affordances ? *Ecological Psychology*, vol. 12, pp. 53–56.
- Gibson J.J. (1977). The theory of affordances. In Shaw R. and Bransford J. (Eds.), *Perceiving, acting, and knowing*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 67–83.
- Gibson J.J. (1979). *The ecological approach to visual perception*. Houghton Mifflin, Boston.
- Girard D. and Ribette J.N. (2001). *OpenSourceJava-Troisième partie : MVC2 avec Struts*. Avril.
- Godet-Bar G., Dupuy-Chessa S. and Nigay L. (2006). Towards a System of Patterns for the Design of Multimodal Interfaces. In *Computer-Aided Design of User Interfaces, Proceedings of 6th International Conference on Computer-Aided Design of User Interfaces CADUI'2006*, Bucharest, Springer-Verlag, Berlin, pp. 27–40.
- Goldberg A. (1984). *Smalltalk-80, The Interactive Programming Environment*. Addison-Wesley Ed.
- Grolaux D., Van Roy P. and Vanderdonck J. (2002). FlexClock: A Plastic Clock Written in Oz with the QtK Toolkit. *Proceedings of 1st International Workshop on Task Models and Diagrams for User Interface Design TAMODIA 2002*, Bucharest, Romania.
- Guittet L. and Pierra G. (1993). Conception modulaire d'une application graphique interactive de conception technique : la notion d'interacteur. In *Proceedings of Interfaces Homme-Machine*, Lyon, pp. 151–156.
- Gustavsson-Christiernin L. and Torgersson O. (2005). Benefits of a multi-layer design in software with multi-user interfaces. In *Proceedings of IASTED International Conference on Software Engineering*, Innsbruck, Autriche.
- Guzelian G., Cauvet C. and Ramadour P. (2004). Conception et reutilisation de composants : une approche par les buts. In *Actes du XXIIème Congrès INFORSID 2004*, Biarritz, mai.
- Hariri M.A., Tabary D. and Kolski C. (2005a). Plastic HCI generation from its abstract model. In *Proc. of IASTED-HCI Int. Conf. on Human-Computer Interaction (Phoenix, USA, November 2005)*, Hamza M.h. (Ed.), ACTA Press, Anaheim, USA, pp. 246–251.
- Hariri M.A., Tabary D. and Kolski C. (2005b). Model Derivation Principles for Specification and Design of Multi-Platform HCI. In *Proc. of Int. Conference on Machine Intelligent ACIDCA-ICMI'2005*, Tozeur, Tunisia, novembre.
- Hariri M.A., Tabary D. and Kolski C. (2006). Démarche en vue de la Génération d'Interfaces Mobiles et Plastiques. In Defude B. and Lecolinet E. (Eds.), *Actes de la troisième conférence UBIMOB2006 Mobilité et Ubiquité (Paris, septembre 2006)*, ACM Conference Proceedings Series, pp. 127–130, septembre.

- Hariri M.A., Lepreux S., Tabary D. and Kolski C. (2007a). Principes d'adaptation d'interfaces homme-machine dans les systèmes d'information en fonction du contexte et de l'utilisateur. INFORSID 2007, Actes des ateliers du XXVème congrès, Perros-Guirec, France, pp. 298–310, mai.
- Hariri M.A., Petersen J., Idoughi D., Kolski C. and Tabary D. (2007b). Plastic presentation of control data in context-awareness environment. Proc. of EAM'07 European Annual Conference on Human-Decision Making and Manual Control. Lyngby, Denmark.
- Hariri M.A., Lepreux S., Tabary D. and Kolski C. (2007c). From Plastic to Plastic-Affordant UI. Workshop E HAMASYT'07, Delft, Netherlands, mars.
- Hassine I., Rieu D., Bounaas F. and Seghrouchni O. (2002a). Towards a reusable business components model. In Workshop on Reuse in Object Oriented Information Systems Design, Montpellier, France. OOIS.
- Hassine I., Rieu D., Bousnaas F. and Seghrouchni O. (2002b). Symphony : un modèle conceptuel de composants métier. In Cauvet C. (Ed.), Revue Ingénierie des Systèmes d'Information (ISI), numéro special "Connaissances métier en ingénierie des systèmes d'information", 7(4), pp. 35–59.
- Hassine I. (2005). Spécification et formalisation des démarches de développement à base de composants métier: la démarche Symphony. Thèse de doctorat, Institut National Polytechnique de Grenoble, décembre.
- Henricksen K. and Indulska J. (2004). A Software Engineering Framework for Context-Aware Pervasive Computing. Proc. International Conf. Pervasive Computing and communications, Florida, USA.
- Henricksen K., Indulska J. and Rakotonirainy A. (2002). Modeling Context Information in Pervasive Computing Systems, pervasive2002, Vol. 2414, Lecture Notes in Computer Science, Springer, pp. 167–180.
- Hess C., Roman M. and Campbell R.H. (2002). Building Applications for Ubiquitous Computing Environments. In Proc. of the First international Conference on Pervasive Computing (Switzerland, August 2002), Mattern F. and Naghshineh M. (Eds.) Lecture Notes In Computer Science, vol. 2414. Springer-Verlag, London, pp. 16–29.
- Hilbert D.M. and Redmiles D.F. (2000). Extracting usability information from user interface events. ACM Comput. Surv., 32, pp. 384–421.
- Hoareau D. and Mahéo Y. (2005). Distribution of a Hierarchical Component in a Non-Connected Environment. In Proceedings of the 31th Euromicro Conference - Component-Based Software Engineering Track (Porto, Portugal September 2005), IEEE CS Press, pp. 143–150.
- Ignizio J.P. (1991). Introduction to Expert Systems: The Development and Implementation of Rule-Based Expert Systems. McGraw-Hill.
- Jacobsen N. E. and John. B.E. (2000). Two Case Studies in Using Cognitive Walkthrough for Interface Evaluation. Carnegie Mellon University, Report No. CMU-CHII-00-100m, mai.
- Johnson R.E. (1997). « Frameworks = Components + Patterns ». Communications of the ACM, vol. 40, pp. 39–42.

- Kern N. and Schiele B. (2003). ContextAware Notification for Wearable Computing. Proceedings of International Symposium on Wearable Computers ISWC'03, New York, USA.
- Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J. and Griswold W. G. (2001). An overview of AspectJ. In proc. of European Conference on Object-Oriented Programming (ECOOP'2001) (Budapest, Hungary, June 2001), Jørgen Lindskov Knudsen (Ed.), Lecture Notes in Computer Science, Springer, vol. 2072, pp. 327–355.
- Kindberg T. and Fox A. (2002). System Software for Ubiquitous Computing. IEEE Pervasive Computing, 1(1), pp. 70–81.
- Klefstad R., Schmidt D.C. and O’Ryan C. (2002). Towards highly configurable real-time object request brokers. In Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, avril - mai.
- Kolski C. and Grislin E. (1998). A review of intelligent human-machine interfaces in the light of the ARCH model. International Journal of Human-Computer Interaction, vol. 10, pp. 193–231.
- Krasner G. E. and Pope S.T. (1988). A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. Journal of Object Oriented Programming, 1(3), pp. 26–49.
- Larman G. (2005). UML2 et les design patterns. Pearson Education, France.
- Laurillau Y. (2002). Conception et réalisation logicielles pour les collecticiels centrés sur l’activité de groupe : le modèle et la plate-forme Clover. Thèse de doctorat, Université Joseph Fourier, Grenoble, septembre.
- Le Pape C. and Gañçarski S. (2004). Fraîcheur et validité de données répliquées dans des environnements transactionnels. Revue des sciences et technologies de l’information, série, Ingénierie des systèmes d’information (RTSI-ISI), 9(5–6), pp. 163–183.
- Lepreux S. and Vanderdonckt J. (2006a). Toward a support of the user interfaces design using composition rules. Proc. of the 6th International Conference on Computer-Aided Design of User Interfaces CADUI'2006, Romania.
- Lepreux S., Vanderdonckt J. and Michotte B. (2006b). Visual Design of User Interfaces by (De)composition. In Proceedings of 13th Int. Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'2006 (Dublin, July 2006), Doherty G. and Blandford A. (Eds.), Springer, Lecture Notes in Computer Science (LNCS), Berlin, pp. 157–170, juillet.
- Lepreux S., Hariri M.A., Rouillard J., Tabary D., Tarby J.C. and Kolski C. (2007). Towards Multimodal User Interfaces Composition based on UsiXML and MBD principles. In Jacko J. (Ed.), Human-Computer Interaction, Part III, HCII 2007, Beijing, Lecture Notes in Computer Science (LNCS) 4552, Springer-Verlag, pp. 134–143.
- Lie H.W. and Bos B. (1999). Cascading Style Sheets, Designing for the Web. Addison Wesley, 2nd edition.
- Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L. and Lopez V. (2004). UsiXML: a Language Supporting Multi-Path Development of User Interfaces. Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-

- DSV-IS'2004, Hamburg, Lecture Notes in Computer Science, Springer-Verlag, Berlin, vol. 3425, pp. 200–220.
- Limbourg Q. and Vanderdonckt J. (2005). Transformational Approach of User Interface with Graph Transformations. In Proceedings of the ACM Int. Conf. on Intelligent User Interfaces jointly organized with ACM Int. Conf. on Computer-Aided Design of User Interfaces, IUI-CADUI 04 (Funchal, January 2004), Jacob R.J., Limbourg Q. and Vanderdonckt J. (Eds.), Kluwer Academics, Dordrecht, 2005.
- Mackay W.E., Velay G., Carter K., Ma C. and Pagani D. (1993). Augmenting Reality: Adding Computational Dimensions to Paper. *Communications of the ACM*, 36(7), pp. 96–97.
- Macrelle-Rosselle M. (2003). Etude des objectifs d'une plate-forme de coopération pour les EIAH. Actes de la conférence EIAH 2003, Strasbourg, pp. 379–390.
- Mahfoudhi A., Abid M. and Abed M. (2005). Towards a user interface generation approach based on object oriented design and task model. *Proceeding of Int. conf. TAMODIA'05*, Gdansk, Poland, pp. 135–142.
- McBryan T. and Gray P.D. (2008). A Model-Based Approach to Supporting Configuration in Ubiquitous Systems. In Proc. of int. Conf. on Design, Specification and Verification of Interactive Systems DSV-IS'2008 (Kingston, Canada, July 16 - 18, 2008), Graham T.C.N. and Palanque P.A (Eds.), Lecture Notes in Computer Science (LNCS) 5136, Springer-Verlag, Berlin, pp. 167-180, 2008.
- McKinley P.K., Sadjadi S.M., Kasten E.P. and Cheng B.H.C. (2004a). A Taxonomy of Compositional Adaptation. Technical Report MSU-CSE-04-17, Department of Computer Science and Engineering, Michigan State University.
- McKinley P.K., Sadjadi S.M., Kasten E.P. and Cheng B.H. (2004b). Composing adaptive software, *IEEE Computer*, 37(7), pp. 56–64.
- Merrick R.A., Wood B. and Krebs W. (2004). Abstract User Interface Markup Language. In *Advances on User Interface Description Languages. Workshop of Advanced Visual Interfaces (AVI) 2004*, Expertise Center for Digital Media.
- Middleton S.E., De Roure D.C. and Shadbolt N.R. (2001). Capturing knowledge of user preferences: ontologies in recommender systems. *Proceedings of the 1st international Conference on Knowledge Capture*, Canada.
- Moha N., Huynh D.L. and Guéhéneuc Y.G. (2006). Une taxonomie et un métamodèle pour la détection des défauts de conception. In *Actes du colloque Langages et Modèles à Objets*, Lavoisier, pp. 201–216.
- Mori G., Paternò F. and Santoro C. (2002). CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Transactions on Software Engineering*, 28(8), pp.797–813.
- Mori G., Paternò F. and Santoro C. (2003). Tool Support for Designing Nomadic Applications. *Proceedings of IUI 2003*, Miami, USA.
- Mori G., Paternò F. and Santoro C. (2004). Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions. *IEEE Transactions on Software Engineering*, 30(8), pp. 507–520.

- Moussa F., Kolski C. and Riahi M. (2006). Analyse des dysfonctionnements des systèmes complexes en amont de la conception des IHM : apports, difficultés, et étude de cas. *Revue d'Interaction Homme-Machine (RIHM)*, 7, pp. 79–111.
- Murthy S.K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4), pp. 345–389.
- Nanard J. (2002). Les patrons dans la conception et la réalisation d'hypermedias, vers une opérationnalisation. *Revue d'Interaction Homme-Machine*, 3(1), pp. 41–78.
- Nielsen J. (1993). *Usability engineering*, Boston, Academic Press.
- Nigay L. (1994). *Conception et Modélisation Logicielle des Systèmes Interactifs : Application aux Interfaces Multimodales*. Thèse de doctorat, Université Joseph Fourier, Grenoble, janvier.
- Norman D.A. (1988). *The psychology of everyday things*. Basic Books, New York.
- Norman D.A. (1994). Les artefacts cognitifs. *Raisons Pratiques. Objets dans l'action*, n°4, pp. 15–34.
- OMG XML (2000). *Metadata Interchange (XML) Specification V1.1*, Specification, Object Management Group.
- Oreizy P., Medvodovic N. and Taylor R.N. (1998). Architecture-Based Runtime Software Evolution. In *Proc. of the Int. Conference on Software Engineering ICSE'98*, Kyoto, pp. 11–15.
- Osuna E., Freund R. and Girosi F. (1997). An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan and E. Wilson (Eds.), *Neural Networks for Signal Processing VII: Proceedings of the 1997 IEEE Workshop*, New York, USA, 276–285. IEEE.
- Ouadou K. (1994). *AMF : Un modèle d'architecture multi-agents multi-facettes pour Interfaces Homme-Machine et les outils associés*. Thèse de doctorat, Ecole Centrale de Lyon.
- Pantic M., Pentland A., Nijholt A. and Huang T. (2006). Human Computing and Machine Understanding of Human Behavior: A Survey. *Proceedings of ACM SIGCHI Eighth International Conference on Multimodal Interfaces (ACM ICMI 2006)*, pp. 239–248.
- Pascoe J. (1998). *Adding Generic Contextual Capabilities to Wearable Computers*. *Proceedings of 2nd International Symposium on Wearable Computers*.
- Paternò F. (1999). *Model Based Design and Evaluation of Interactive Applications*. Springer Verlag, Berlin.
- Paternò F. and Santoro C. (2002). One Model, Many Interfaces. In Kolski C. and Vanderdonck J. (Eds.), *Computer-Aided Design of User Interfaces III*. Kluwer Academic Publishers, Dordrecht, pp. 143–154.
- Patry G. (1999). *Contribution à la conception du dialogue Homme-Machine dans les applications graphiques interactives de conception technique: le système GIPSE*. Thèse de doctorat, Université de Poitiers.
- Petersen J. and May M. (2003). Scale Transformations and Information Presentation in Supervisory Control. In *Proceedings of the 22nd Conference on Human Decision Making and Control (University of Linkping, juin 2003)*, Lützhft M. (Ed.), pp. 75–85.

- Petersen J. and May M. (2006). Scale transformations and information presentation in supervisory control. *Int. J. Hum.-Comput. Stud.*, 64(5), pp. 405–419.
- Pfaff G.E. (1985). *User Interface Management Systems*, Eurographics Seminars, Springer Verlag.
- Pinna-Dery A.M., Fierstone J. and Picard E. (2003). Component model and programming: a first step to manage human computer interaction adaptation. In *The 5th International Symposium on Human-Computer Interaction with Mobile Devices and Services*, Lecture Notes in Computer Science, vol. 2795, pp. 456–460. Chittaro L. (Ed.), Udine, Italie, septembre.
- Pinna-Dery A.M., Fierstone J., Riveill M. and Picard E. (2004). *User Interface: a Technical Component in Component-Based Models in Response to Human Computer Interaction Adaptation*. Rapport de recherche I3S/RR-2004-06-FR, Laboratoire I3S, Université de Nice Sophia, février.
- Pottie G. and Kaiser W. (2000). Wireless Integrated Network Sensors. *Comm. ACM*, 43(5), pp. 51–58.
- Puerta, A. and Eisenstein, J. (1998). Interactively Mapping Task Model to Interfaces in Mobi-D. *Proceedings of the Eurographics Workshop on Design, Specification and Validation of Interactive Systems (DSV-IS'98)* (Abingdon, UK, June 1998), pp. 261–274.
- Puerta A. and Eisenstein J. (2002). XIML: A Common Representation for Interaction Data. In *IUI'02 6th International Conference on Intelligent User Interfaces*, pp. 214–215.
- Quinlan J.R. (1979). Discovering Rules by Induction From Large Collections of Examples. In Michie D. (Ed.), *Expert Systems in the Micro Electronic Age*, Edinburgh University Press, pp. 168–201.
- Quinlan J.R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, California, USA.
- Rasmussen J., and Vicente K. (1989). Coping with human errors through system design: implications for ecological interface design. *International Journal of Man-Machine Studies*, vol. 31, pp. 517–534.
- Redmond B. and Cahill V. (2002). Supporting unanticipated dynamic adaptation of application behaviour. In *Proceedings of the 16th European Conference on Object-Oriented Programming (Malaga, Spain 2002)*, Lecture Notes in Computer Science, Springer-Verlag, vol. 2374, juin.
- Riahi M. (2004). Contribution à l'élaboration d'une méthodologie de spécification, de vérification et de génération semi-automatique d'interfaces homme-machine : Application à l'outil Ergo-Conceptor+. Thèse de doctorat, LAMIH, Université de Valenciennes.
- Riehle D. (1997). Composite design patterns. In *Proceedings of the 12th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (Atlanta USA, October 1997)*, Berman A.M. (Ed.), OOPSLA '97, ACM Press, New York, NY, pp. 218–228.

- Rosa N.S., Cunha P.R.F. and Justo G.R.R. (2002). ProcessNFL: A Language for Describing Non-Functional Properties. In Proceedings of the 35th Hawaii International Conference on System Sciences, IEEE Press.
- Ross D.T. (1977). Structured Analysis (SA): A Language for Communicating Ideas. IEEE Transactions on Software Engineering, 3(1), pp. 16–34.
- Rouillard J. (2003). Plastic ML and its toolkit. HCI International 2003, Heraklion, Crete, Greece, Vol. 4, pp. 612–616.
- Russell S.J. and Norvig P. (2002). Artificial Intelligence: A Modern Approach (2nd Edition). Prentice Hall.
- Savidis A. (2005). Dynamic Software Assembly for Automatic Deployment-oriented Adaptation. Electr. Notes Theor. Comput. Sci., 127(3), pp. 207–211.
- Savidis A. and Stephanidis C. (2006). Automated user interface engineering with a pattern reflecting programming language. Automated Software Engineering, 13(2), pp. 303–339.
- Schilit B. and Theimer M. (1994). Disseminating active map information to mobile hosts. IEEE Network, 8(5), pp. 22–32.
- Schmidt A. (1999). Advanced Interaction in Context, Proc. 1st Int’l Symp. Handheld and Ubiquitous Computing HUC 99 (Karlsruhe Germany, 1999), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1707(1), pp. 89–101.
- Schmucker K. (1986). MacApp : an application framework. Byte Magazine, 11(8), pp. 189–193.
- Seffah A. and Gaffar A. (2007). Model-based user interface engineering with design patterns. Journal System Software, 80(8), pp.1408–1422.
- Simpson N. (1999). Managing the use of style guides in an organisational setting: practical lessons in ensuring UI consistency. Interacting with Computers, 11(3), pp. 323–351.
- Sining D., Gaffar A., Reichart D., Forbrig P. and Seffah A. (2004). Patterns in Model-Based Engineering. proceedings of Computer-Aided Design of User Interfaces CADUI’2004. Island of Madeira, Portugal, pp. 197–210.
- Sire S. (2004). Point de vue sur la plasticité, Journées thématiques de l’AS Plasticité du RTP 16 IHM, Namur, Belgium, août.
- Sottet J.S., Calvary G. and Favre J.M. (2005). Towards Model-Driven Engineering of Plastic User Interfaces. Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI’05) held in conjunction with the ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MODELS’05), Montego Bay, Jamaica.
- Stafford J.A. and Wallnau K. (2002). Building reliable component-based software systems. In Crnkovic I. and Larsson M. (Eds.), Component Composition and Integration, Artech House Publishers, pp. 179–191.
- Stanciulescu A., Limbourg Q., Vanderdonckt J., Michotte B. and Montero F. (2005). A Transformational Approach for Multimodal Web User Interfaces based on UsiXML, Proc. of 7th Int. Conf. on Multimodal Interfaces ICMI’2005 (Trento, October 2005), ACM Press, New York, 2005, pp. 259–266.

- Stevens S.S. (1946). On the theory of scales and measurement. *Science*, 103(2684), pp. 677–680.
- Sutcliffe A. (2001). On the Effective Use and Reuse of HCI Knowledge, *ACM Transactions on Computer-Human Interaction*, 7(2), pp. 197–221.
- Szekely P. (1996). Retrospective and challenge for Model Based Interface Development. *Proceedings of Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96)* (Namur, Belgium, June 1996), pp. 1–27.
- Szekely P.A., Sukaviriya P.N., Castells P., Muthukumarasamy J. and Salcher E. (1995). Declarative interface models for user interface construction tools: the MASTERMIND approach. In Bass L. and Unger C. (Eds.), *Engineering for Human-Computer Interaction*, Chapman & Hall, London, pp. 120–150.
- Szyperski C. (1998). *Component Software. Beyond Object-Oriented Programming* - Addison-Wesley / ACM Press, 1998.
- Szyperski C. (2002). *Component Software*, 2nd ed., Addison-Wesley.
- Tabary D. (2001). Contribution à TOOD, une méthode à base de modèles pour la spécification et la conception des systèmes interactifs. Thèse de doctorat, Université de Valenciennes, décembre.
- Tam R.C., Maulsby D. and Puerta A. R. (1998). U-TEL: a tool for eliciting user task models from domain experts. In *Proceedings of the 3rd international Conference on intelligent User interfaces IUI '98* (San Francisco, janvier 1998). ACM, New York, pp. 77–80.
- Tarby J.C. and Barthet M.F. (1996). The Diane+ editor. *Proceedings of CADUI'96*, Namur, Belgium.
- Tarby J.C. (2004). One Goal, Many Tasks, Many Devices: From Abstract User Task Specification to User Interfaces. In *the Handbook of Task Analysis for Human-Computer Interaction*, Diaper D. and Stanton N.A. (Eds.), Lawrence Erlbaum Associates, pp. 531–550.
- Tarpin F. (1997). Travail coopératif synchrone assisté par ordinateur : Approche AMF-C. Thèse de doctorat, Ecole Centrale de Lyon.
- Tarpin F. and David B. (1999). AMF : un modèle d'architecture multi-agents multi-facettes. *Techniques et Sciences Informatiques*, 18(5), mai, pp. 555–586.
- Tarpin F. (2006). *Interaction Homme-Machine Adaptative*. Habilitation à Diriger des Recherches, Université Lyon I et INSA de Lyon, décembre.
- Thevenin D. and Coutaz J. (1999). Plasticity of User Interfaces: Framework and Research Agenda. In *Proc. of 7th IFIP Int. Conference on HCI Interact'99*, Edinburgh, Scotland, pp. 110–117.
- Thevenin D. (2001). *Adaptation en Interaction Homme-Machine : Le cas de la Plasticité*. Thèse de doctorat, Université Joseph Fourier, Grenoble.
- Tsia J.J.P. and Zhang D. (2005). *Machine Learning Applications In Software Engineering*. World Scientific.
- UIMS. (1992). The UIMS Workshop Tool Developers. A metamodel for the runtime architecture of an interactive system. *SIGCHI Bulletin*, 24(1), pp. 32–37.

- Van den Bergh J. and Coninx K. (2004). Model-Based Design of Context-Sensitive Interactive Applications: a Discussion of Notations. In Proceedings of 3rd International Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, Czech Republic, November 2004), ACM, pp. 43–50.
- Van Welie M. and Van Der Veer G.C. (2003). Pattern Languages in Interaction Design: Structure and Organization. In Proceedings of Interact '03 (Zürich, Switzerland, September 2003), Rauterberg M., Menozzi M. and Wesson J. (Eds.), IOS Press, Amsterdam, The Netherlands, pp. 527–534.
- Vanderdonckt J. (1994). Guide ergonomique de la présentation des applications hautement interactives, Presses Universitaires de Namur, Belgique.
- Vanderdonckt J. (1997). Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive. Thèse des Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, juillet.
- Vanderdonckt J. (1999a). Advice-giving systems for selecting interaction objects. In Proc. of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edinburg, September 1999), Paton N.W. and Griffiths T. (Eds.), IEEE Computer Society Press, Los Alamitos, pp. 152–157.
- Vanderdonckt J. (1999b). Development Milestones towards a Tool for Working with Guidelines. *Interacting with Computers*, 12 (2), pp. 81–118.
- Vanderdonckt J., Bouillon L. and Souchon N. (2001). Flexible Reverse Engineering of Web Pages with VAQUITA. In Proceedings of IEEE 8th Working Conference on Reverse Engineering WCRE'2001 (Stuttgart, octobre 2001), IEEE Press, Los Alamitos, pp. 241–248.
- Vanderdonckt J., Limbourg Q., Michotte B., Bouillon L., Trevisan D. and Florins M. (2004). UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In Proc. of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, juillet.
- Vanderdonckt J. and Coyette A. (2006). Vers un prototypage des interfaces graphiques incluant vraiment l'utilisateur final. In Proc. of 10ème Colloque Int. sur l'Ergonomie et l'Informatique Avancée ERGO-IA'2006 (Biarritz, October 2006), Brangier E., Kolski C. and Ruault J.R. (Eds.), Ecole Supérieure des Technologies Industrielles Avancées (ESTIA/ILS), Bidart.
- Vestal S. (1998). MetaH Programmer's Manuel version 1.27. Honeywell Technology Center.
- Vicente, K. (1999). Cognitive work analysis: : Toward Safe, Productive, and Healthy Computer-Based Work. Lawrence Erlbaum Associates, Mahwah, New Jersey.
- Weiser M. (1991). The computer for the 21st century. *Scientific American*, 265 (3), pp. 94–104.
- Welch I. and Stroud R.J. (2000). Kava - A Reflective Java Based on Bytecode Rewriting. In Cazzola W., Stroud R.J. and Tisato F. (Eds.), *Reflection and Software Engineering, Lecture Notes in Computer Science*, Heidelberg, Germany, Springer-Verlag, vol. 1826, pp. 157–169.
- Wirfs-Brock R., Wilkerson B. and Weiner L. (1990). Designing Object-Oriented Software, Prentice-Hall, Englewood Cliffs, New Jersey.

Woods W. (1970). Transition Network Grammars for Natural Language Analysis. *Communications of the ACM*, 13(10), pp. 591–606.

Annexe A

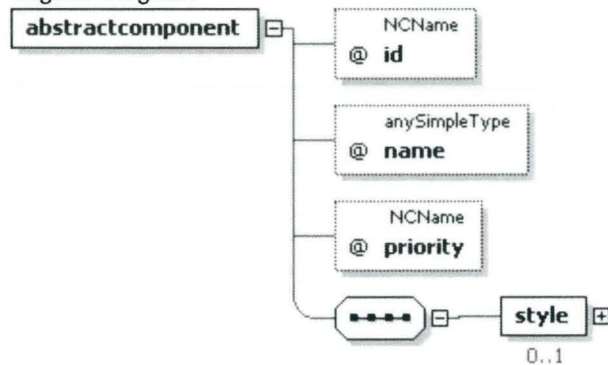
DTD du modèle AMXML

Global Declarations

Element: **abstractcomponent**

Name	abstractcomponent
Used by (from the same schema document)	Element abstractcontainer
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



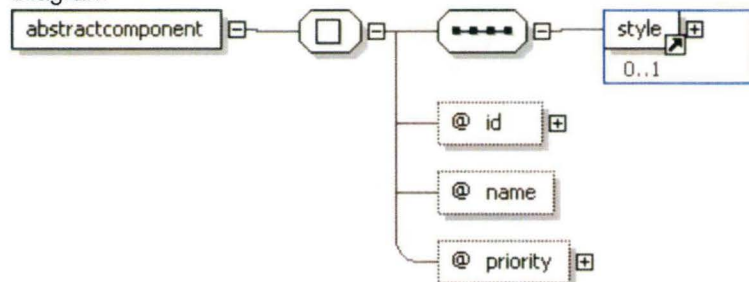
XML Instance Representation

```

<abstractcomponent
id=" xs:NCName [1]"
name="anySimpleType [1]"
priority=" xs:NCName [0..1]">
<style> ... </style> [0..1]
</abstractcomponent>

```

Diagram



Schema Component Representation

```

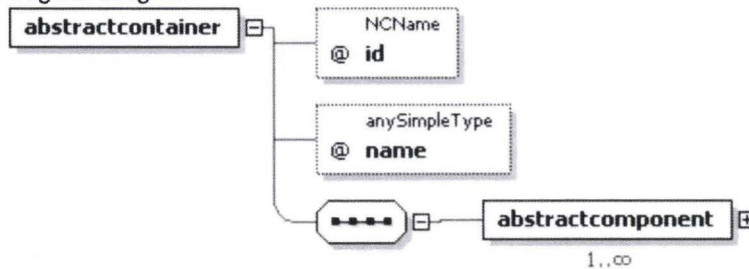
<xs:element name="abstractcomponent">
<xs:complexType>
<xs:sequence>
<xs:element ref=" style " minOccurs="0"/>
</xs:sequence>
<xs:attribute name="id" type=" xs:NCName " use="required"/>
<xs:attribute name="name" use="required"/>
<xs:attribute name="priority" type=" xs:NCName "/>
</xs:complexType>
</xs:element>

```

Element: **abstractcontainer**

Name	abstractcontainer
Used by (from the same schema document)	Element auiModel
Type	Locally-defined complex type
Nilable	no
Abstract	no

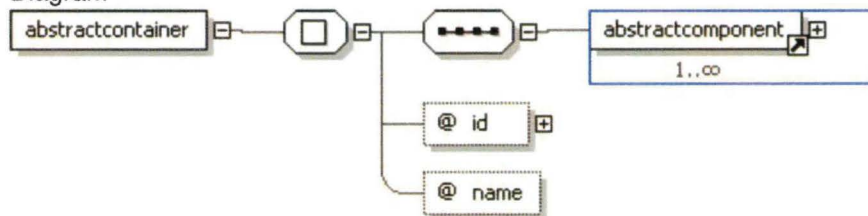
Logical Diagram



XML Instance Representation

```
<abstractcontainer
id=" xs:NCName [1]"
name="anySimpleType [1]">
<abstractcomponent> ... </abstractcomponent> [1..*]
</abstractcontainer>
```

Diagram



Schema Component Representation

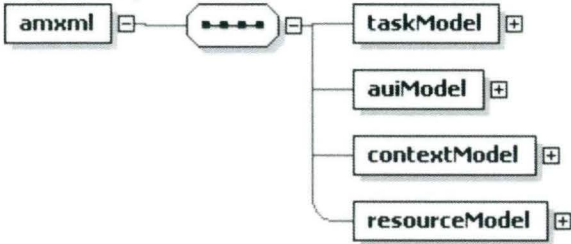
```
<xs:element name="abstractcontainer">
<xs:complexType>
<xs:sequence>
<xs:element ref=" abstractcomponent " maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type=" xs:NCName " use="required"/>
<xs:attribute name="name" use="required"/>
</xs:complexType>
</xs:element>
```

Element: **amxml**

Name	amxml
Type	Locally-defined complex type
Nilable	no

Abstract no

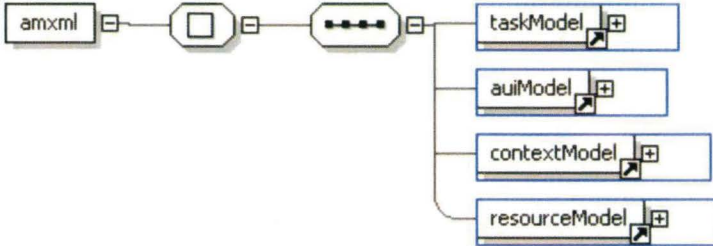
Logical Diagram



XML Instance Representation

```
<amxml>
<taskModel> ... </taskModel> [1]
<auiModel> ... </auiModel> [1]
<contextModel> ... </contextModel> [1]
<resourceModel> ... </resourceModel> [1]
</amxml>
```

Diagram



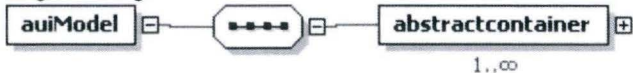
Schema Component Representation

```
<xs:element name="amxml">
<xs:complexType>
<xs:sequence>
<xs:element ref=" taskModel "/>
<xs:element ref=" auiModel "/>
<xs:element ref=" contextModel "/>
<xs:element ref=" resourceModel "/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **auiModel**

Name	auiModel
Used by (from the same schema document)	Element amxml
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



1..∞

XML Instance Representation

```
<auiModel>
<abstractcontainer> ... </abstractcontainer> [1..*]
</auiModel>
```

Diagram



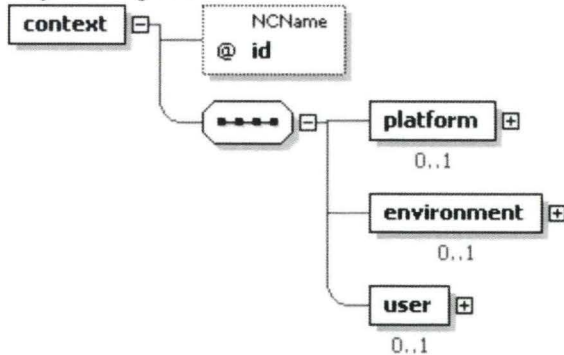
Schema Component Representation

```
<xs:element name="auiModel">
<xs:complexType>
<xs:sequence>
<xs:element ref=" abstractcontainer " maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **context**

Name	context
Used by (from the same schema document)	Element contextModel
Type	Locally-defined complex type
Nilable	no
Abstract	no

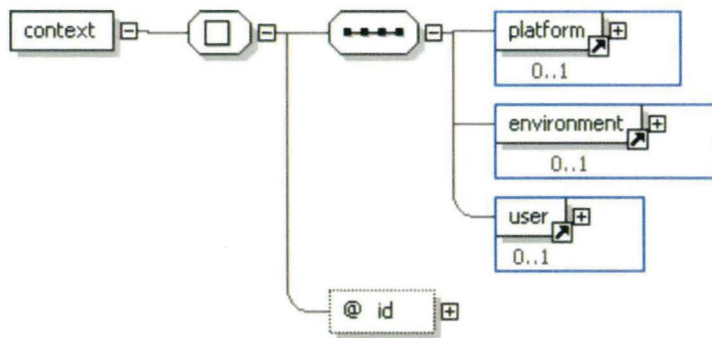
Logical Diagram



XML Instance Representation

```
<context
id=" xs:NCName [1]">
<platform> ... </platform> [0..1]
<environment> ... </environment> [0..1]
<user> ... </user> [0..1]
</context>
```

Diagram



Schema Component Representation

```

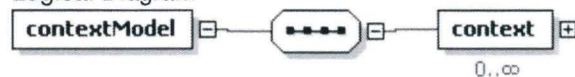
<xs:element name="context">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref=" platform " minOccurs="0"/>
      <xs:element ref=" environment " minOccurs="0"/>
      <xs:element ref=" user " minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type=" xs:NCName " use="required"/>
  </xs:complexType>
</xs:element>

```

Element: contextModel

Name	contextModel
Used by (from the same schema document)	Element amxml
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



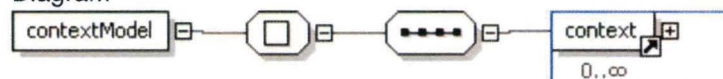
XML Instance Representation

```

<contextModel>
  <context> ... </context> [0..*]
</contextModel>

```

Diagram



Schema Component Representation

```

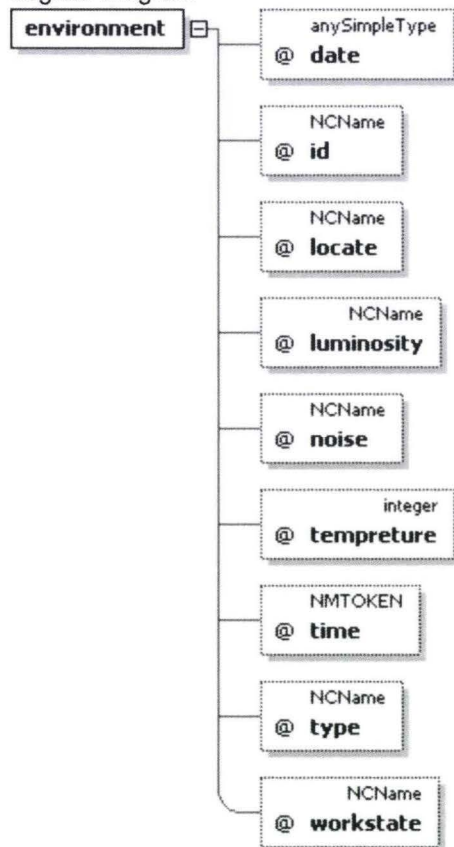
<xs:element name="contextModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref=" context " minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Element: **environment**

Name	environment
Used by (from the same schema document)	Element context
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



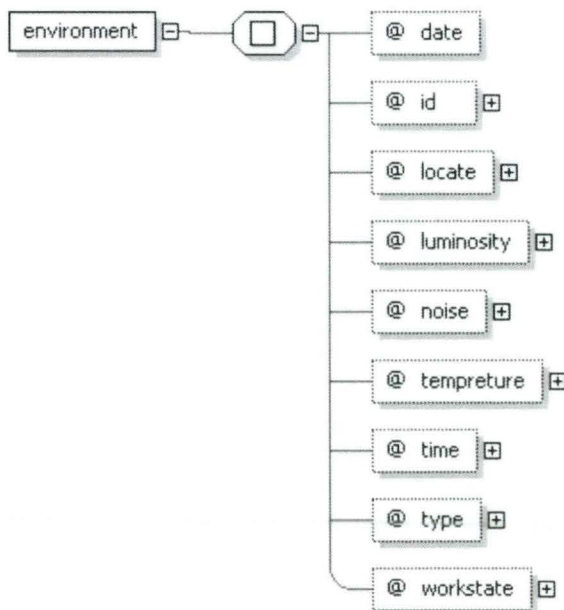
XML Instance Representation

```

<environment
date="anySimpleType [0..1]"
id=" xs:NCName [1]"
locate=" xs:NCName [0..1]"
luminosity=" xs:NCName [0..1]"
noise=" xs:NCName [0..1]"
tempreture=" xs:integer [0..1]"
time=" xs:NMTOKEN [0..1]"
type=" xs:NCName [0..1]"
workstate=" xs:NCName [0..1]"/>

```

Diagram



Schema Component Representation

```

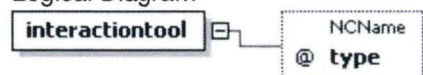
<xs:element name="environment">
  <xs:complexType>
    <xs:attribute name="date"/>
    <xs:attribute name="id" type=" xs:NCName " use="required"/>
    <xs:attribute name="locate" type=" xs:NCName "/>
    <xs:attribute name="luminosity" type=" xs:NCName "/>
    <xs:attribute name="noise" type=" xs:NCName "/>
    <xs:attribute name="tempreture" type=" xs:integer "/>
    <xs:attribute name="time" type=" xs:NMTOKEN "/>
    <xs:attribute name="type" type=" xs:NCName "/>
    <xs:attribute name="workstate" type=" xs:NCName "/>
  </xs:complexType>
</xs:element>

```

Element: **interactiontool**

Name	interactiontool
Used by (from the same schema document)	Element platform
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation

```

<interactiontool
  type=" xs:NCName [1]"/>

```

Diagram



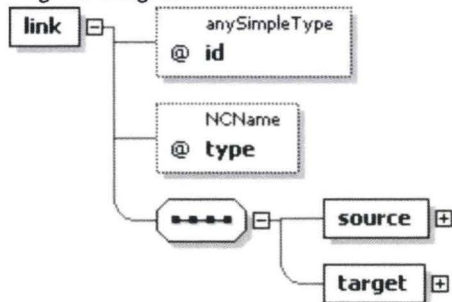
Schema Component Representation

```
<xs:element name="interactiontool">
<xs:complexType>
<xs:attribute name="type" type=" xs:NCName " use="required"/>
</xs:complexType>
</xs:element>
```

Element: **link**

Name	link
Used by (from the same schema document)	Element relationships
Type	Locally-defined complex type
Nilable	no
Abstract	no

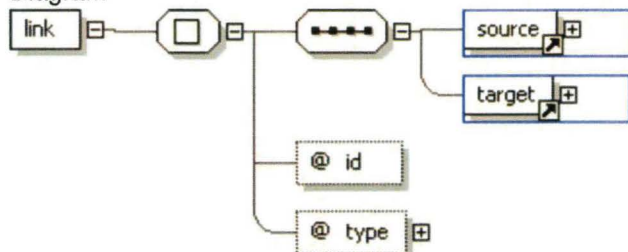
Logical Diagram



XML Instance Representation

```
<link
id="anySimpleType [1]"
type=" xs:NCName [1]">
<source> ... </source> [1]
<target> ... </target> [1]
</link>
```

Diagram



Schema Component Representation

```
<xs:element name="link">
<xs:complexType>
<xs:sequence>
<xs:element ref=" source "/>
<xs:element ref=" target "/>

```

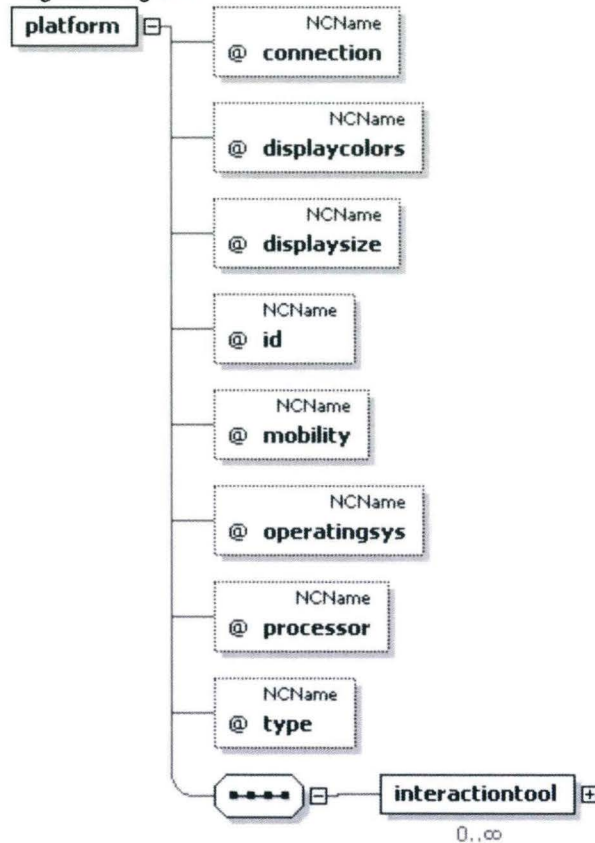
```

</xs:sequence>
<xs:attribute name="id" use="required"/>
<xs:attribute name="type" type=" xs:NCName " use="required"/>
</xs:complexType>
</xs:element>

```

Element: platform

Name	platform
Used by (from the same schema document)	Element context
Type	Locally-defined complex type
<u>Nilable</u>	no
<u>Abstract</u>	no

Logical Diagram

XML Instance Representation

```

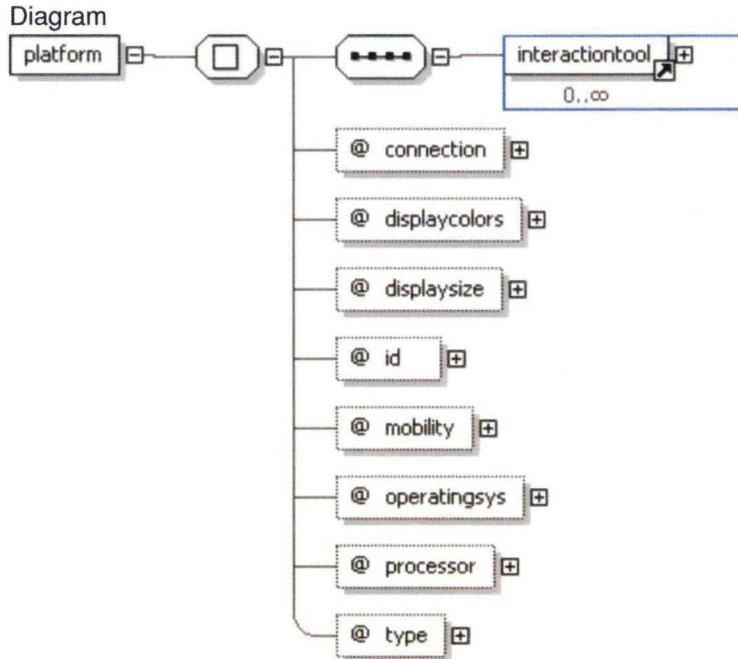
<platform
connection=" xs:NCName [0..1]"
displaycolors=" xs:NCName [0..1]"
displaysize=" xs:NCName [0..1]"
id=" xs:NCName [1]"
mobility=" xs:NCName [0..1]"
operatingsys=" xs:NCName [0..1]"
processor=" xs:NCName [0..1]"

```

```

type=" xs:NCName [0..1]">
<interactiontool> ... </interactiontool> [0..*]
</platform>
Diagram

```



Schema Component Representation

```

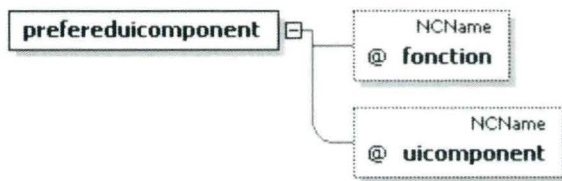
<xs:element name="platform">
<xs:complexType>
<xs:sequence>
<xs:element ref=" interactiontool " minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="connection" type=" xs:NCName "/>
<xs:attribute name="displaycolors" type=" xs:NCName "/>
<xs:attribute name="displaysize" type=" xs:NCName "/>
<xs:attribute name="id" type=" xs:NCName " use="required"/>
<xs:attribute name="mobility" type=" xs:NCName "/>
<xs:attribute name="operatingsys" type=" xs:NCName "/>
<xs:attribute name="processor" type=" xs:NCName "/>
<xs:attribute name="type" type=" xs:NCName "/>
</xs:complexType>
</xs:element>

```

Element: **prefereduicomponent**

Name	prefereduicomponent
Used by (from the same schema document)	Element user
Type	Locally-defined complex type
Nilable	no
Abstract	no

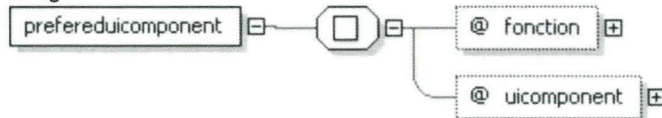
Logical Diagram



XML Instance Representation

```
<prefereducicomponent
  fonction=" xs:NCName [1]"
  uicomponent=" xs:NCName [1]"/>
```

Diagram



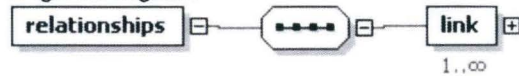
Schema Component Representation

```
<xs:element name="prefereducicomponent">
  <xs:complexType>
    <xs:attribute name="fonction" type=" xs:NCName " use="required"/>
    <xs:attribute name="uicomponent" type=" xs:NCName " use="required"/>
  </xs:complexType>
</xs:element>
```

Element: **relationships**

Name	relationships
Used by (from the same schema document)	Element taskModel
Type	Locally-defined complex type
Nilable	no
Abstract	no

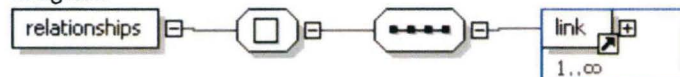
Logical Diagram



XML Instance Representation

```
<relationships>
  <link> ... </link> [1..*]
</relationships>
```

Diagram



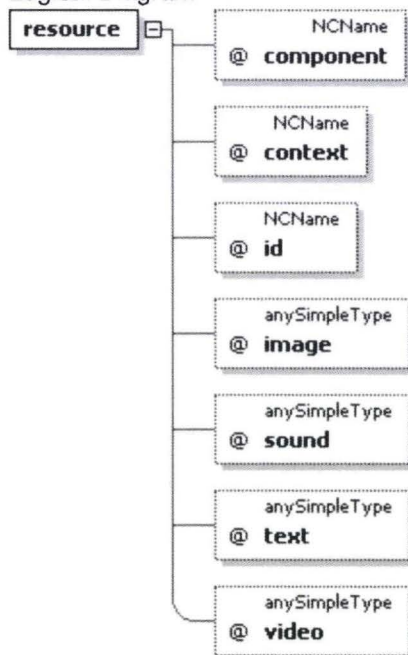
Schema Component Representation

```
<xs:element name="relationships">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref=" link " maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```


Element: **resource**

Name	resource
Used by (from the same schema document)	Element resourceModel
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram

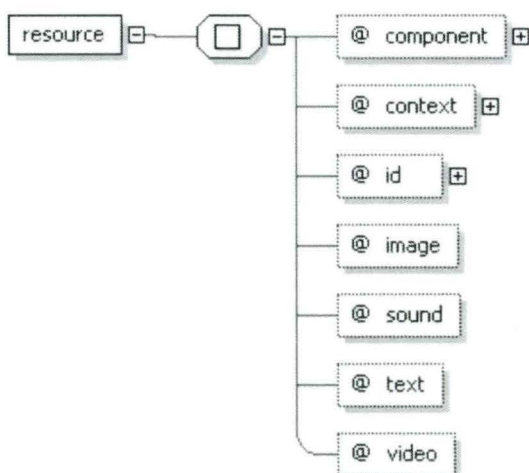


XML Instance Representation

```

<resource
  component=" xs:NCName [1]"
  context=" xs:NCName [1]"
  id=" xs:NCName [1]"
  image="anySimpleType [0..1]"
  sound="anySimpleType [0..1]"
  text="anySimpleType [0..1]"
  video="anySimpleType [0..1]" />
  
```

Diagram



Schema Component Representation

```

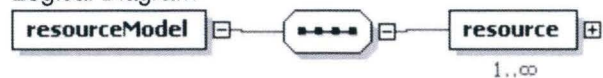
<xs:element name="resource">
  <xs:complexType>
    <xs:attribute name="component" type=" xs:NCName " use="required"/>
    <xs:attribute name="context" type=" xs:NCName " use="required"/>
    <xs:attribute name="id" type=" xs:NCName " use="required"/>
    <xs:attribute name="image"/>
    <xs:attribute name="sound"/>
    <xs:attribute name="text"/>
    <xs:attribute name="video"/>
  </xs:complexType>
</xs:element>

```

Element: resourceModel

Name	resourceModel
Used by (from the same schema document)	Element amxml
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



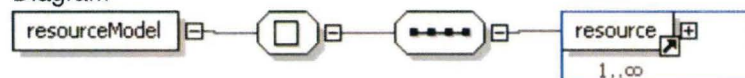
XML Instance Representation

```

<resourceModel>
  <resource> ... </resource> [1..*]
</resourceModel>

```

Diagram



Schema Component Representation

```

<xs:element name="resourceModel">
  <xs:complexType>

```

```

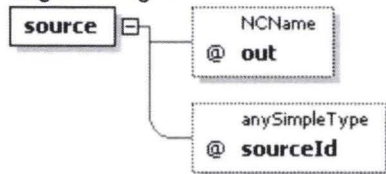
<xs:sequence>
<xs:element ref=" resource " maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

```

Element: source

Name	source
Used by (from the same schema document)	Element link
Type	Locally-defined complex type
<u>Nillable</u>	no
<u>Abstract</u>	no

Logical Diagram



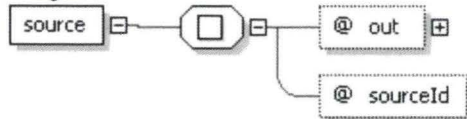
XML Instance Representation

```

<source
out=" xs:NCName [1]"
sourceId="anySimpleType [1]"/>

```

Diagram



Schema Component Representation

```

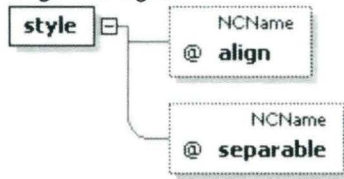
<xs:element name="source">
<xs:complexType>
<xs:attribute name="out" type=" xs:NCName " use="required"/>
<xs:attribute name="sourceId" use="required"/>
</xs:complexType>
</xs:element>

```

Element: style

Name	style
Used by (from the same schema document)	Element abstractcomponent
Type	Locally-defined complex type
<u>Nillable</u>	no
<u>Abstract</u>	no

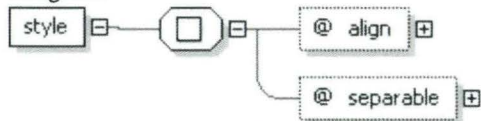
Logical Diagram



XML Instance Representation

```
<style
align=" xs:NCName [1]"
separable=" xs:NCName [1]"/>
```

Diagram



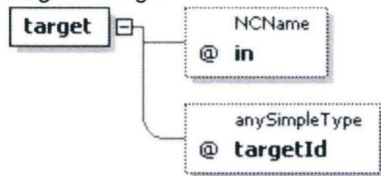
Schema Component Representation

```
<xs:element name="style">
<xs:complexType>
<xs:attribute name="align" type=" xs:NCName " use="required"/>
<xs:attribute name="separable" type=" xs:NCName " use="required"/>
</xs:complexType>
</xs:element>
```

Element: **target**

Name	target
Used by (from the same schema document)	Element link
Type	Locally-defined complex type
Nilable	no
Abstract	no

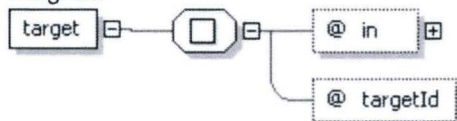
Logical Diagram



XML Instance Representation

```
<target
in=" xs:NCName [1]"
targetId="anySimpleType [1]"/>
```

Diagram



Schema Component Representation

```
<xs:element name="target">
<xs:complexType>
```



```

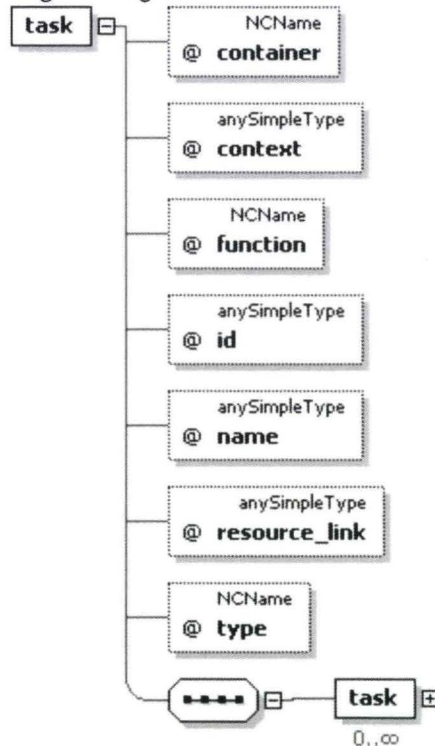
<xs:attribute name="in" type=" xs:NCName " use="required"/>
<xs:attribute name="targetId" use="required"/>
</xs:complexType>
</xs:element>

```

Element: **task**

Name	task
Used by (from the same schema document)	Element taskModel , Element task
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



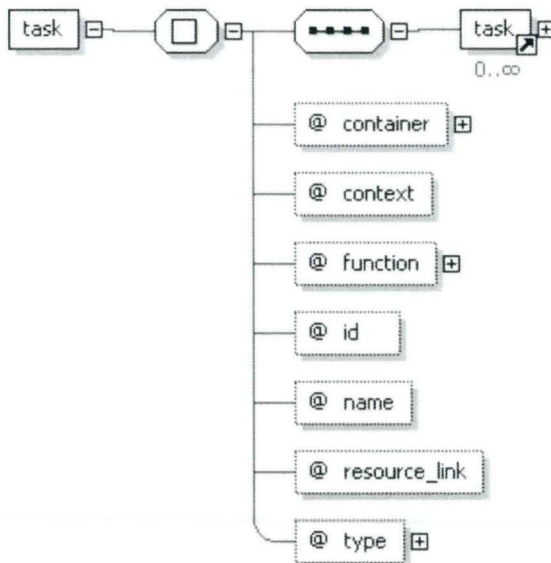
XML Instance Representation

```

<task
  container=" xs:NCName [0..1]"
  context="anySimpleType [0..1]"
  function=" xs:NCName [0..1]"
  id="anySimpleType [1]"
  name="anySimpleType [1]"
  resource_link="anySimpleType [0..1]"
  type=" xs:NCName [1]">
  <task> ... </task> [0..*]
</task>

```

Diagram



Schema Component Representation

```

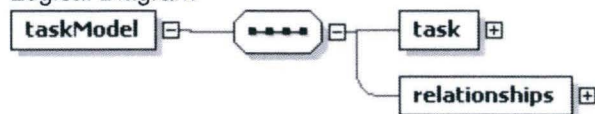
<xs:element name="task">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref=" task " minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="container" type=" xs:NCName "/>
    <xs:attribute name="context"/>
    <xs:attribute name="function" type=" xs:NCName "/>
    <xs:attribute name="id" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="resource_link"/>
    <xs:attribute name="type" type=" xs:NCName " use="required"/>
  </xs:complexType>
</xs:element>

```

Element: **taskModel**

Name	taskModel
Used by (from the same schema document)	Element amxml
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation

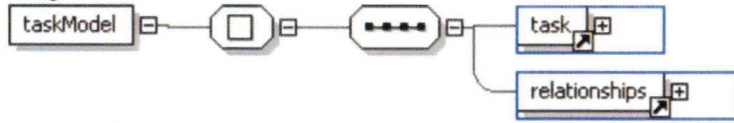
```

<taskModel>
  <task> ... </task> [1]

```

```
<relationships> ... </relationships> [1]
</taskModel>
```

Diagram



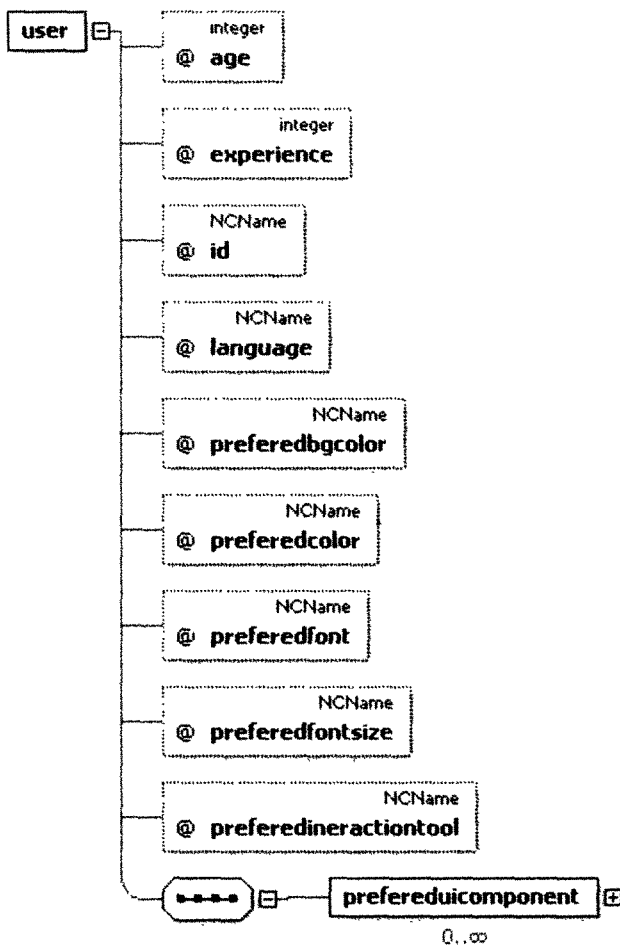
Schema Component Representation

```
<xs:element name="taskModel">
<xs:complexType>
<xs:sequence>
<xs:element ref=" task "/>
<xs:element ref=" relationships "/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **user**

Name	user
Used by (from the same schema document)	Element context
Type	Locally-defined complex type
<u>Nilable</u>	no
<u>Abstract</u>	no

Logical Diagram



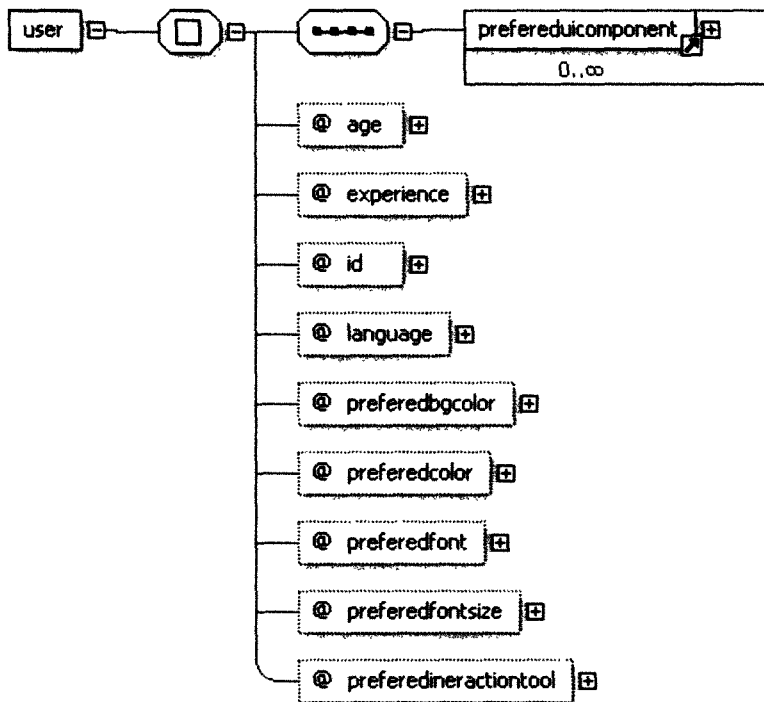
XML Instance Representation

```

<user
age=" xs:integer [0..1]"
experience=" xs:integer [0..1]"
id=" xs:NCName [1]"
language=" xs:NCName [0..1]"
preferredbgcolor=" xs:NCName [0..1]"
preferredcolor=" xs:NCName [0..1]"
preferredfont=" xs:NCName [0..1]"
preferredfontsize=" xs:NCName [0..1]"
preferredinactiontool=" xs:NCName [0..1]">
<preferredcomponent> ... </preferredcomponent> [0..*]
</user>

```

Diagram



Schema Component Representation

```

<xs:element name="user">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="prefereduicomponent" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="age" type="xs:integer"/>
    <xs:attribute name="experience" type="xs:integer"/>
    <xs:attribute name="id" type="xs:NCName" use="required"/>
    <xs:attribute name="language" type="xs:NCName"/>
    <xs:attribute name="preferredbgcolor" type="xs:NCName"/>
    <xs:attribute name="preferredcolor" type="xs:NCName"/>
    <xs:attribute name="preferredfont" type="xs:NCName"/>
    <xs:attribute name="preferredfontsize" type="xs:NCName"/>
    <xs:attribute name="preferredineractiontool" type="xs:NCName"/>
  </xs:complexType>
</xs:element>

```


Annexe B

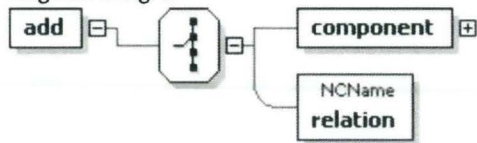
DTD du gabarit-patron de conception

Global Declarations

Element: **add**

Name	add
Used by (from the same schema document)	Element reaction , Element repeat , Element case
Type	Locally-defined complex type
<u>Nilable</u>	no
<u>Abstract</u>	no

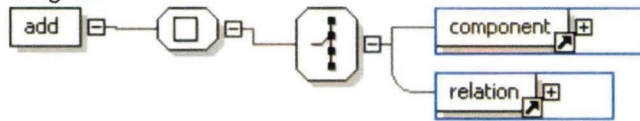
Logical Diagram



XML Instance Representation

```
<add>
Start Choice [1]
<component> ... </component> [1]
<relation> ... </relation> [1]
End Choice
</add>
```

Diagram



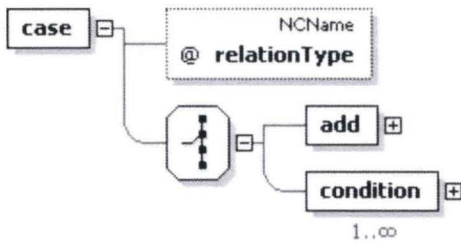
Schema Component Representation

```
<xs:element name="add">
<xs:complexType>
<xs:choice>
<xs:element ref=" component "/>
<xs:element ref=" relation "/>
</xs:choice>
</xs:complexType>
</xs:element>
```

Element: **case**

Name	case
Used by (from the same schema document)	Element switch , Element component
Type	Locally-defined complex type
<u>Nilable</u>	no
<u>Abstract</u>	no

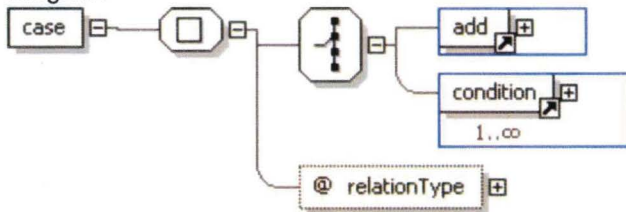
Logical Diagram



XML Instance Representation

```
<case relationType=" xs:NCName [0..1]">
Start Choice [1]
<add> ... </add> [1]
<condition> ... </condition> [1..*]
End Choice
</case>
```

Diagram



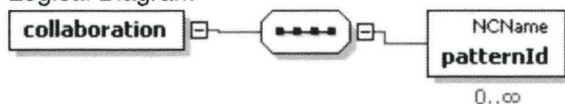
Schema Component Representation

```
<xs:element name="case">
<xs:complexType>
<xs:choice>
<xs:element ref=" add "/>
<xs:element ref=" condition " maxOccurs="unbounded"/>
</xs:choice>
<xs:attribute name="relationType" type=" xs:NCName "/>
</xs:complexType>
</xs:element>
```

Element: **collaboration**

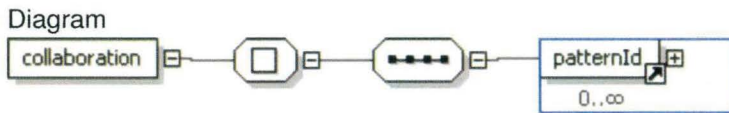
Name	collaboration
Used by (from the same schema document)	Element pattern
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation

```
<collaboration>
<!-- Mixed content -->
<patternId> ... </patternId> [0..*]
</collaboration>
```



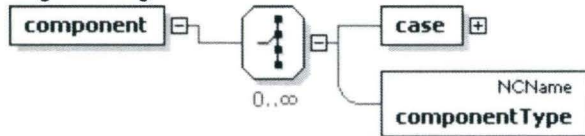
Schema Component Representation

```
<xs:element name="collaboration">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref=" patternId " minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **component**

Name	component
Used by (from the same schema document)	Element delete , Element repeat , Element add , Element have
Type	Locally-defined complex type
Nilable	no
Abstract	no

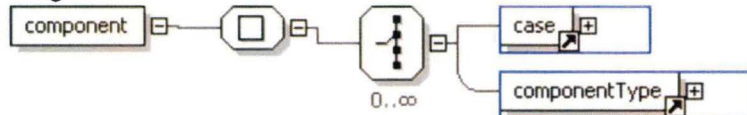
Logical Diagram



XML Instance Representation

```
<component>
<!-- Mixed content -->
Start Choice [0..*]
<case> ... </case> [1]
<componentType> ... </componentType> [1]
End Choice
</component>
```

Diagram



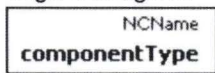
Schema Component Representation

```
<xs:element name="component">
<xs:complexType mixed="true">
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element ref=" case "/>
<xs:element ref=" componentType "/>
</xs:choice>
</xs:complexType>
</xs:element>
```

Element: **componentType**

Name	componentType
Used by (from the same schema document)	Element component
Type	xs:NCName
<u>Nilable</u>	no
<u>Abstract</u>	no

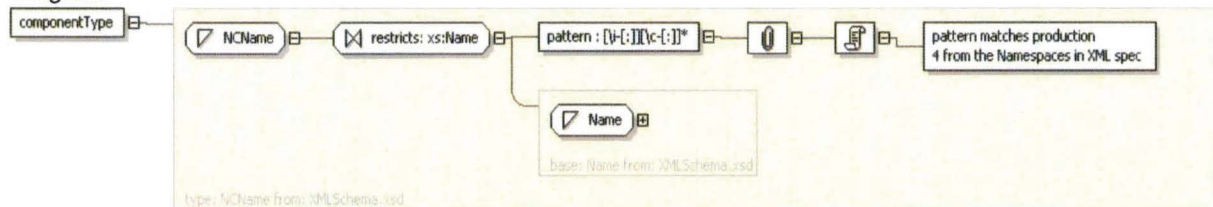
Logical Diagram



XML Instance Representation

```
<componentType> xs:NCName </componentType>
```

Diagram



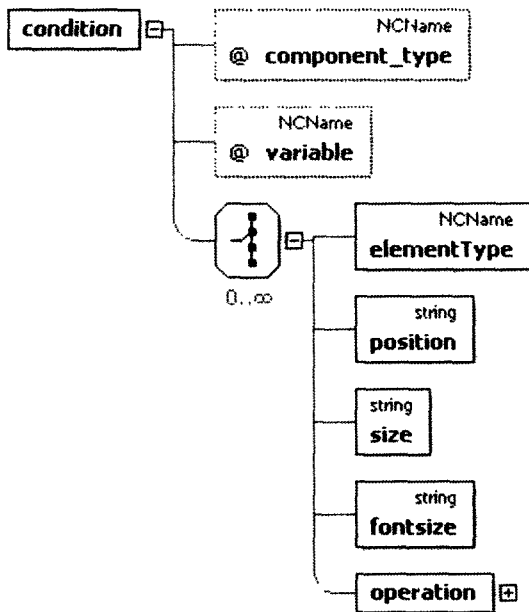
Schema Component Representation

```
<xs:element name="componentType" type=" xs:NCName "/>
```

Element: **condition**

Name	condition
Used by (from the same schema document)	Element solution , Element case
Type	Locally-defined complex type
<u>Nilable</u>	no
<u>Abstract</u>	no

Logical Diagram



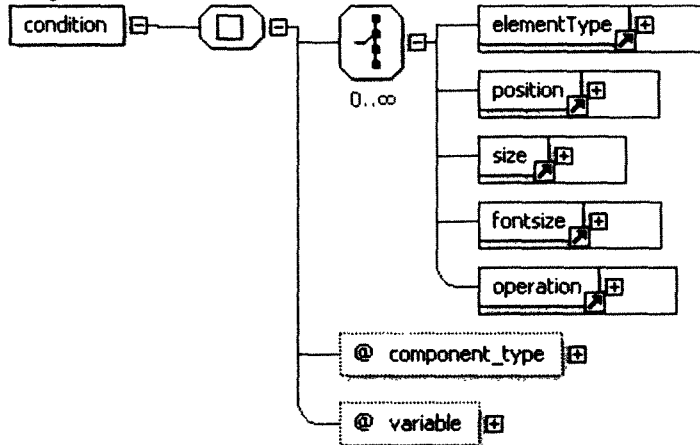
XML Instance Representation

```

<condition
  component_type=" xs:NCName [0..1]"
  variable=" xs:NCName [0..1]">
  <!-- Mixed content -->
  Start Choice [0..*]
  <elementType> ... </elementType> [1]
  <position> ... </position> [1]
  <size> ... </size> [1]
  <fontsize> ... </fontsize> [1]
  <operation> ... </operation> [1]
  End Choice
</condition>

```

Diagram



Schema Component Representation

```

<xs:element name="condition">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref=" elementType "/>
      <xs:element ref=" position "/>
      <xs:element ref=" size "/>

```



```

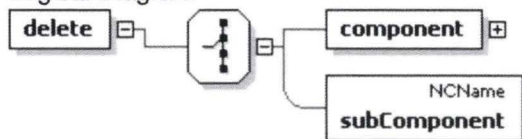
<xs:element ref=" fontsize "/>
<xs:element ref=" operation "/>
</xs:choice>
<xs:attribute name="component_type" type=" xs:NCName "/>
<xs:attribute name="variable" type=" xs:NCName "/>
</xs:complexType>
</xs:element>

```

Element: delete

Name	delete
Used by (from the same schema document)	Element reaction , Element repeat
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



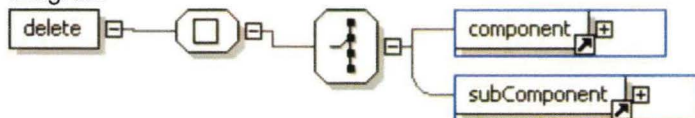
XML Instance Representation

```

<delete>
Start Choice [1]
<component> ... </component> [1]
<subComponent> ... </subComponent> [1]
End Choice
</delete>

```

Diagram



Schema Component Representation

```

<xs:element name="delete">
<xs:complexType>
<xs:choice>
<xs:element ref=" component "/>
<xs:element ref=" subComponent "/>
</xs:choice>
</xs:complexType>
</xs:element>

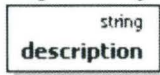
```

Element: description

Name	description
Used by (from the same schema document)	Element pattern

Type	xs:string
Nilable	no
Abstract	no

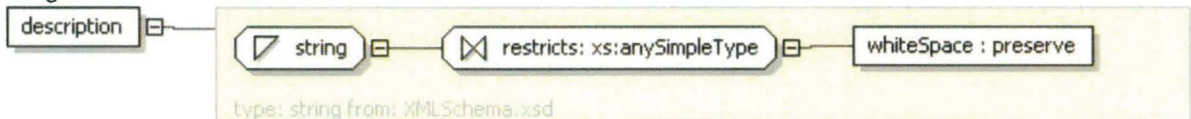
Logical Diagram



XML Instance Representation

<description> xs:string </description>

Diagram



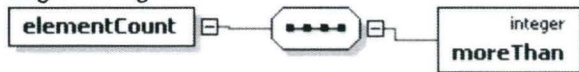
Schema Component Representation

<xs:element name="description" type="xs:string" />

Element: **elementCount**

Name	elementCount
Used by (from the same schema document)	Element subElements
Type	Locally-defined complex type
Nilable	no
Abstract	no

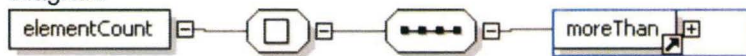
Logical Diagram



XML Instance Representation

<elementCount>
<moreThan> ... </moreThan> [1]
</elementCount>

Diagram



Schema Component Representation

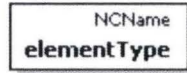
```
<xs:element name="elementCount">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref=" moreThan" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Element: **elementType**

Name	elementType
Used by (from the same schema)	Element condition , Element operation , Element subElements

document)	
Type	xs:NCName
Nilable	no
Abstract	no

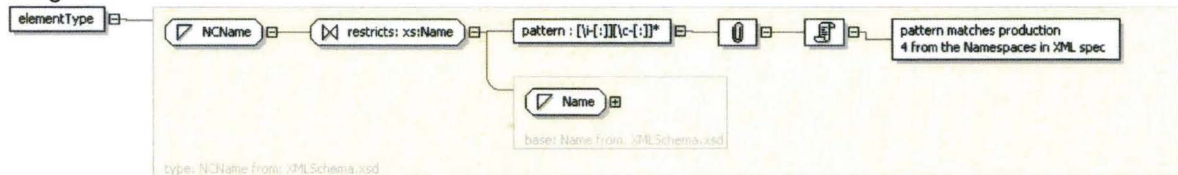
Logical Diagram



XML Instance Representation

`<elementType> xs:NCName </elementType>`

Diagram



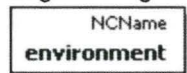
Schema Component Representation

`<xs:element name="elementType" type=" xs:NCName "/>`

Element: **environment**

Name	environment
Used by (from the same schema document)	Element environments
Type	xs:NCName
Nilable	no
Abstract	no

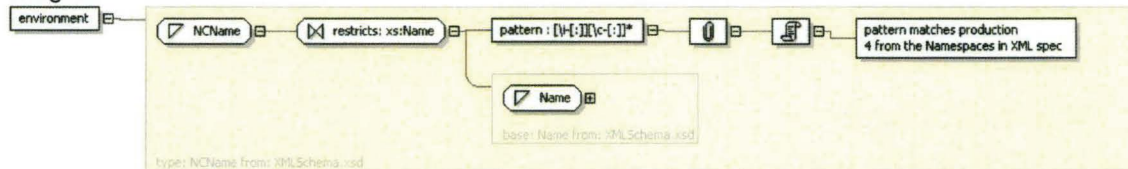
Logical Diagram



XML Instance Representation

`<environment> xs:NCName </environment>`

Diagram



Schema Component Representation

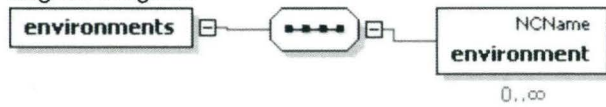
`<xs:element name="environment" type=" xs:NCName "/>`

Element: **environments**

Name	environments
Used by (from the same schema document)	Element useContext

Type	Locally-defined complex type
<u>Nillable</u>	no
<u>Abstract</u>	no

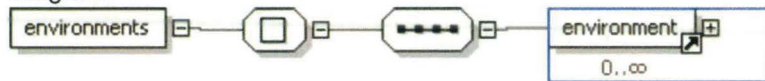
Logical Diagram



XML Instance Representation

```
<environments>
<!-- Mixed content -->
<environment> ... </environment> [0..*]
</environments>
```

Diagram



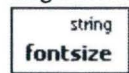
Schema Component Representation

```
<xs:element name="environments">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref="environment" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **fontsize**

Name	fontsize
Used by (from the same schema document)	Element condition
Type	xs:string
<u>Nillable</u>	no
<u>Abstract</u>	no

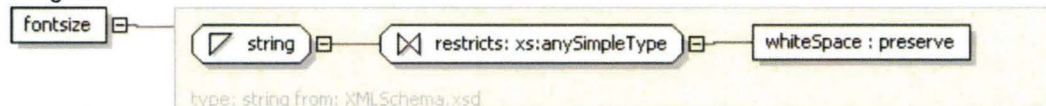
Logical Diagram



XML Instance Representation

```
<fontsize> xs:string </fontsize>
```

Diagram



Schema Component Representation

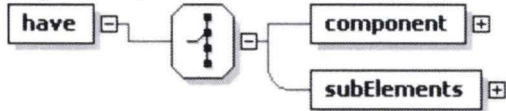
```
<xs:element name="fontsize" type="xs:string"/>
```

Element: **have**

Name	have
-------------	------

Used by (from the same document)	Element operation
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation

<have>

Start Choice [1]

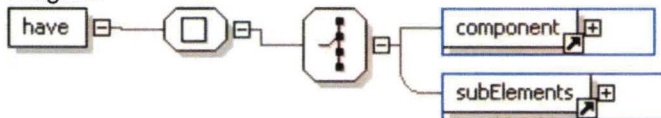
<component> ... </component> [1]

<subElements> ... </subElements> [1]

End Choice

</have>

Diagram



Schema Component Representation

<xs:element name="have">

<xs:complexType>

<xs:choice>

<xs:element ref="component"/>

<xs:element ref="subElements"/>

</xs:choice>

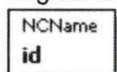
</xs:complexType>

</xs:element>

Element: **id**

Name	id
Used by (from the same document)	Element pattern
Type	xs:NCName
Nilable	no
Abstract	no

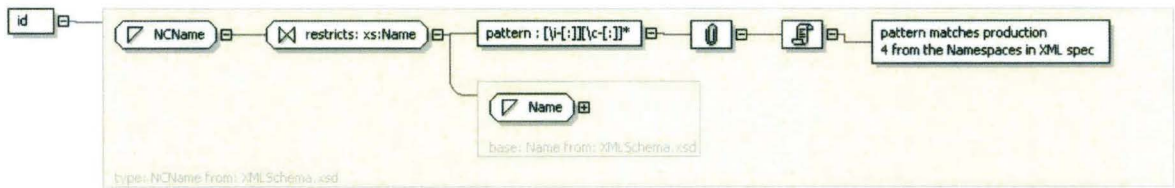
Logical Diagram



XML Instance Representation

<id> xs:NCName </id>

Diagram

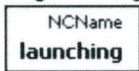


Schema Component Representation
`<xs:element name="id" type=" xs:NCName "/>`

Element: **launching**

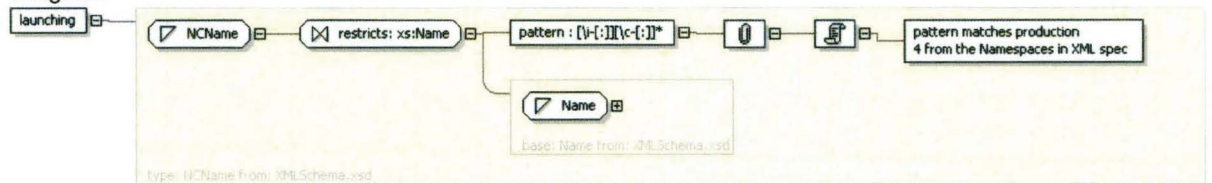
Name	launching
Used by (from the same schema document)	Element pattern
Type	xs:NCName
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation
`<launching> xs:NCName </launching>`

Diagram

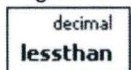


Schema Component Representation
`<xs:element name="launching" type=" xs:NCName "/>`

Element: **lessthan**

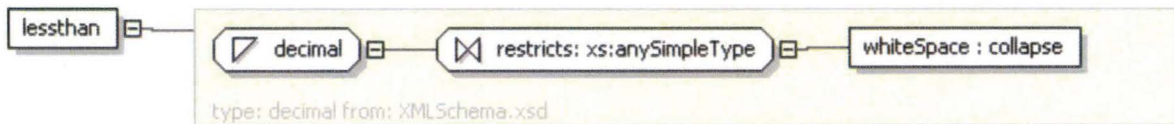
Name	lessthan
Used by (from the same schema document)	Element operation
Type	xs:decimal
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation
`<lessthan> xs:decimal </lessthan>`

Diagram



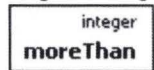
Schema Component Representation

`<xs:element name="lessthan" type="xs:decimal"/>`

Element: **moreThan**

Name	moreThan
Used by (from the same schema document)	Element elementCount
Type	xs:integer
<u>Nillable</u>	no
<u>Abstract</u>	no

Logical Diagram



XML Instance Representation

`<moreThan> xs:integer </moreThan>`

Diagram



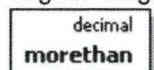
Schema Component Representation

`<xs:element name="moreThan" type="xs:integer"/>`

Element: **morethan**

Name	morethan
Used by (from the same schema document)	Element operation
Type	xs:decimal
<u>Nillable</u>	no
<u>Abstract</u>	no

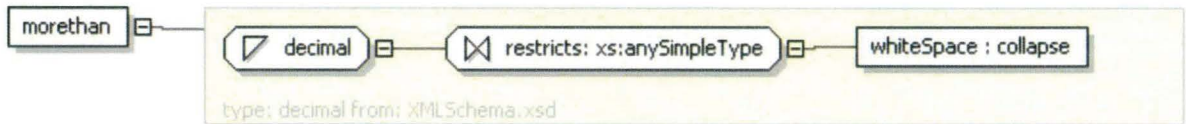
Logical Diagram



XML Instance Representation

`<morethan> xs:decimal </morethan>`

Diagram



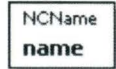
Schema Component Representation

```
<xs:element name="morethan" type="xs:decimal" />
```

Element: **name**

Name	name
Used by (from the same schema document)	Element pattern
Type	xs:NCName
Niltable	no
Abstract	no

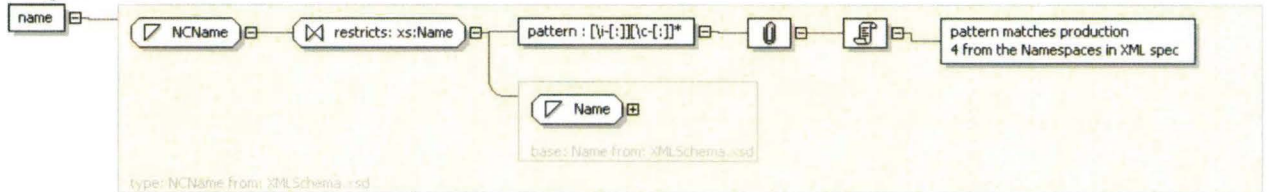
Logical Diagram



XML Instance Representation

```
<name> xs:NCName </name>
```

Diagram



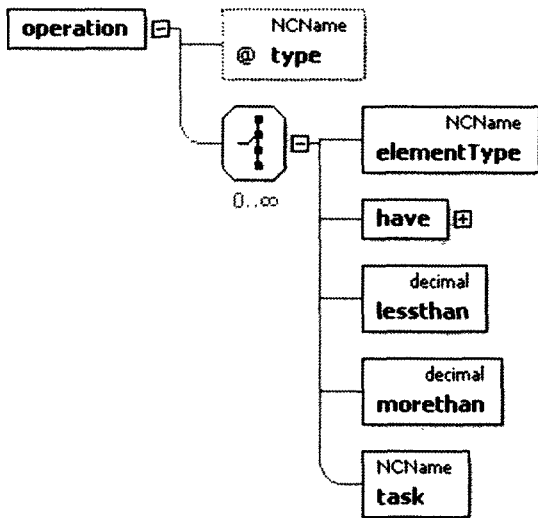
Schema Component Representation

```
<xs:element name="name" type="xs:NCName" />
```

Element: **operation**

Name	operation
Used by (from the same schema document)	Element condition
Type	Locally-defined complex type
Niltable	no
Abstract	no

Logical Diagram



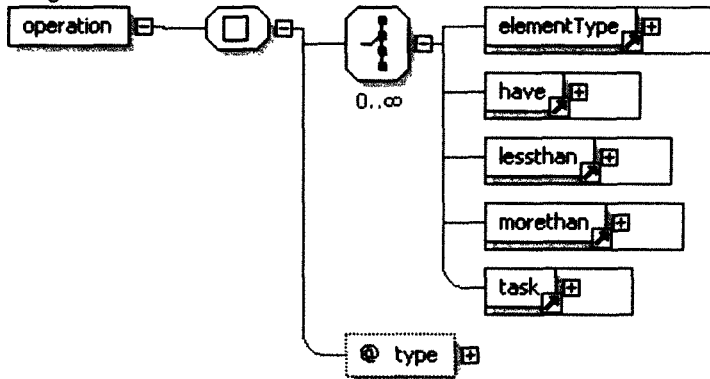
XML Instance Representation

```

<operation type=" xs:NCName [1]">
<!-- Mixed content -->
Start Choice [0..*]
<elementType> ... </elementType> [1]
<have> ... </have> [1]
<lessthan> ... </lessthan> [1]
<morethan> ... </morethan> [1]
<task> ... </task> [1]
End Choice
</operation>

```

Diagram



Schema Component Representation

```

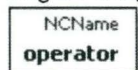
<xs:element name="operation">
<xs:complexType mixed="true">
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element ref=" elementType "/>
<xs:element ref=" have "/>
<xs:element ref=" lessthan "/>
<xs:element ref=" morethan "/>
<xs:element ref=" task "/>
</xs:choice>
<xs:attribute name="type" type=" xs:NCName " use="required"/>
</xs:complexType>
</xs:element>

```

Element: **operator**

Name	operator
Used by (from the same schema document)	Element subElements
Type	xs:NCName
Nilable	no
Abstract	no

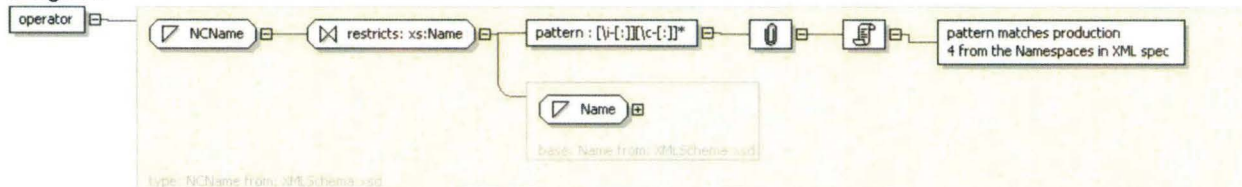
Logical Diagram



XML Instance Representation

```
<operator> xs:NCName </operator>
```

Diagram



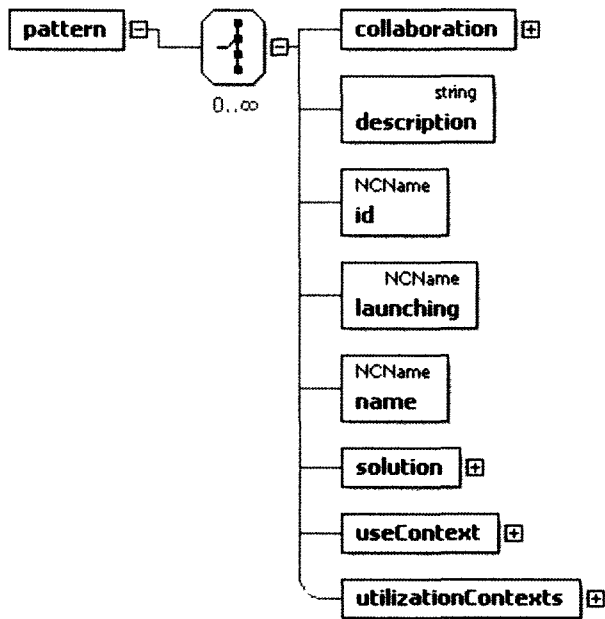
Schema Component Representation

```
<xs:element name="operator" type=" xs:NCName "/>
```

Element: **pattern**

Name	pattern
Used by (from the same schema document)	Element patternLibrary
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



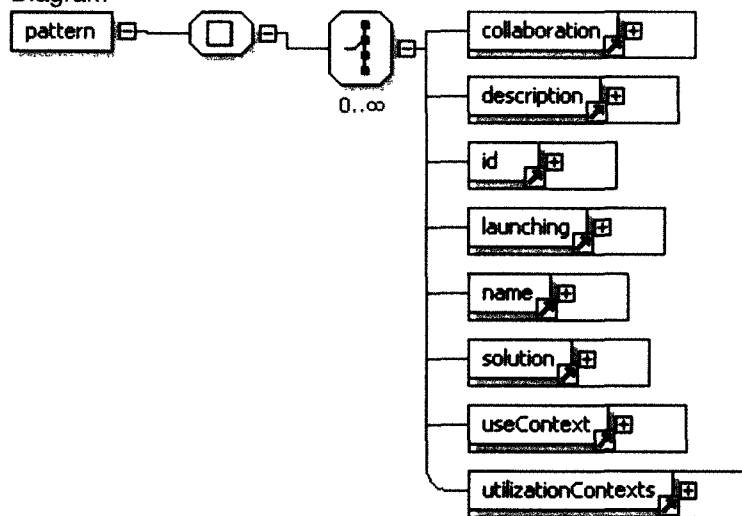
XML Instance Representation

```

<pattern>
<!-- Mixed content -->
Start Choice [0..*]
<collaboration> ... </collaboration> [1]
<description> ... </description> [1]
<id> ... </id> [1]
<launching> ... </launching> [1]
<name> ... </name> [1]
<solution> ... </solution> [1]
<useContext> ... </useContext> [1]
<utilizationContexts> ... </utilizationContexts> [1]
End Choice
</pattern>

```

Diagram



Schema Component Representation

```

<xs:element name="pattern">
<xs:complexType mixed="true">
<xs:choice minOccurs="0" maxOccurs="unbounded">

```

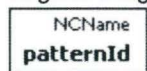
```

<xs:element ref=" collaboration "/>
<xs:element ref=" description "/>
<xs:element ref=" id "/>
<xs:element ref=" launching "/>
<xs:element ref=" name "/>
<xs:element ref=" solution "/>
<xs:element ref=" useContext "/>
<xs:element ref=" utilizationContexts "/>
</xs:choice>
</xs:complexType>
</xs:element>
    
```

Element: **patternId**

Name	patternId
Used by (from the same schema document)	Element collaboration
Type	xs:NCName
<u>Nilable</u>	no
<u>Abstract</u>	no

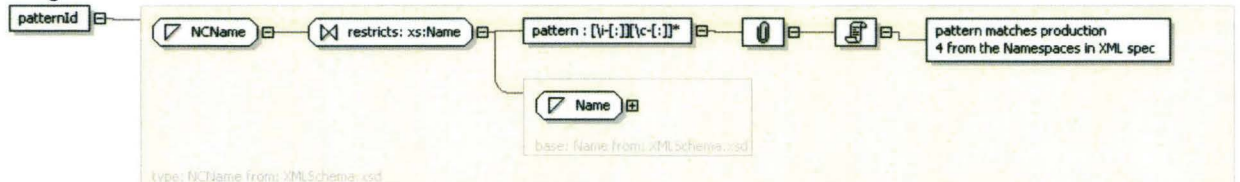
Logical Diagram



XML Instance Representation

```
<patternId> xs:NCName </patternId>
```

Diagram



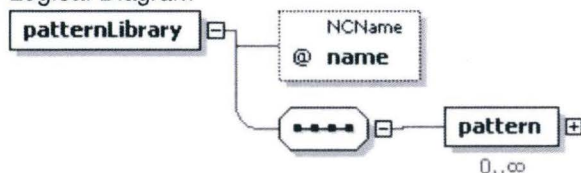
Schema Component Representation

```
<xs:element name="patternId" type=" xs:NCName "/>
```

Element: **patternLibrary**

Name	patternLibrary
Type	Locally-defined complex type
<u>Nilable</u>	no
<u>Abstract</u>	no

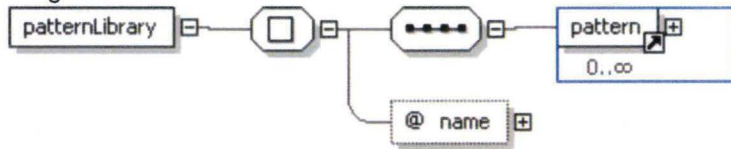
Logical Diagram



XML Instance Representation


```
<patternLibrary
name=" xs:NCName [1]">
<!-- Mixed content -->
<pattern> ... </pattern> [0..*]
</patternLibrary>
```

Diagram



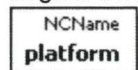
Schema Component Representation

```
<xs:element name="patternLibrary">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref=" pattern " minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type=" xs:NCName " use="required"/>
</xs:complexType>
</xs:element>
```

Element: **platform**

Name	platform
Used by (from the same schema document)	Element platforms
Type	xs:NCName
Nilable	no
Abstract	no

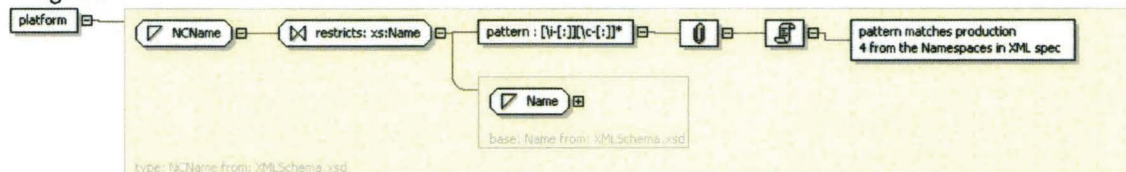
Logical Diagram



XML Instance Representation

```
<platform> xs:NCName </platform>
```

Diagram



Schema Component Representation

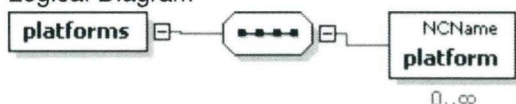
```
<xs:element name="platform" type=" xs:NCName "/>
```

Element: **platforms**

Name	platforms
Used by (from the same schema document)	Element useContext

Type	Locally-defined complex type
Nilable	no
Abstract	no

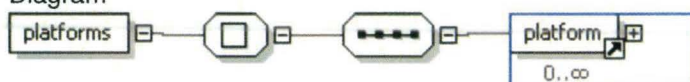
Logical Diagram



XML Instance Representation

```
<platforms>
<!-- Mixed content -->
<platform> ... </platform> [0..*]
</platforms>
```

Diagram



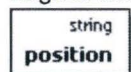
Schema Component Representation

```
<xs:element name="platforms">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref=" platform " minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **position**

Name	position
Used by (from the same schema document)	Element window , Element condition
Type	xs:string
Nilable	no
Abstract	no

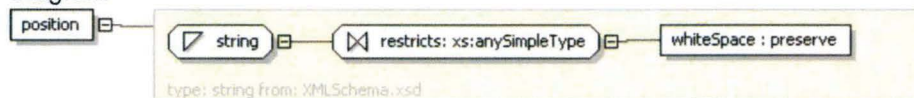
Logical Diagram



XML Instance Representation

```
<position> xs:string </position>
```

Diagram



Schema Component Representation

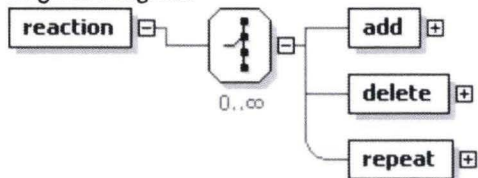
```
<xs:element name="position" type=" xs:string "/>
```

Element: **reaction**

Name	reaction
-------------	----------

Used by (from the same schema document)	Element solution
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



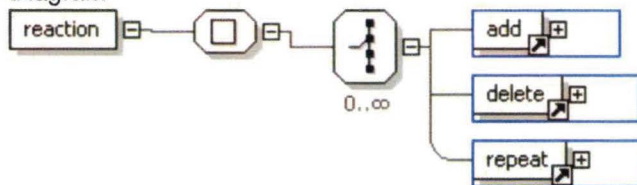
XML Instance Representation

```

<reaction>
<!-- Mixed content -->
Start Choice [0..*]
<add> ... </add> [1]
<delete> ... </delete> [1]
<repeat> ... </repeat> [1]
End Choice
</reaction>

```

Diagram



Schema Component Representation

```

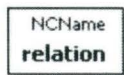
<xs:element name="reaction">
<xs:complexType mixed="true">
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element ref=" add "/>
<xs:element ref=" delete "/>
<xs:element ref=" repeat "/>
</xs:choice>
</xs:complexType>
</xs:element>

```

Element: **relation**

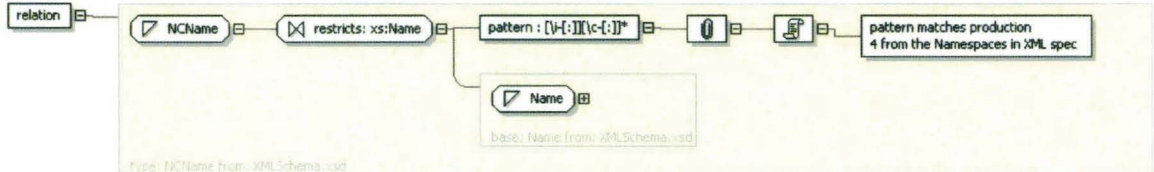
Name	relation
Used by (from the same schema document)	Element add
Type	xs:NCName
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation
`<relation> xs:NCName </relation>`

Diagram

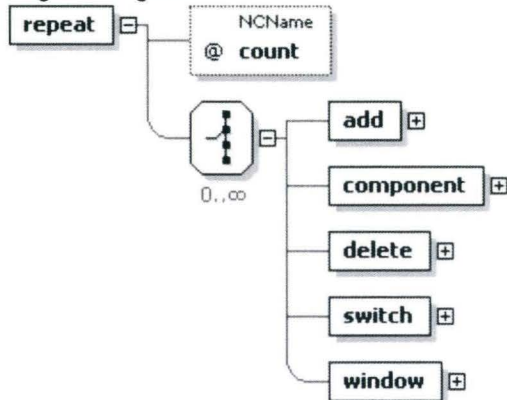


Schema Component Representation
`<xs:element name="relation" type=" xs:NCName "/>`

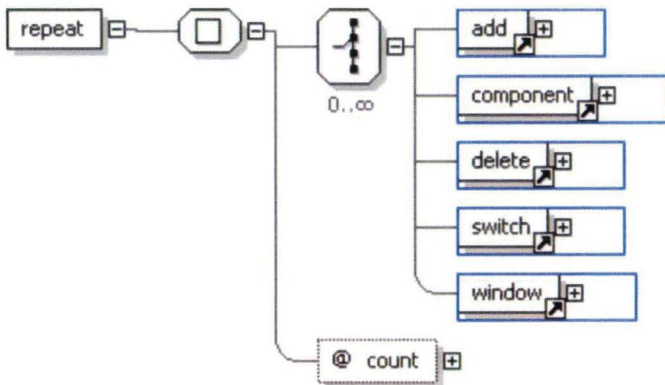
Element: **repeat**

Name	repeat
Used by (from the same schema document)	Element solution , Element reaction , Element window
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram



XML Instance Representation
`<repeat
 count=" xs:NCName [0..1]">
 <!-- Mixed content -->
 Start Choice [0..*]
 <add> ... </add> [1]
 <component> ... </component> [1]
 <delete> ... </delete> [1]
 <switch> ... </switch> [1]
 <window> ... </window> [1]
 End Choice
</repeat>`
 Diagram



Schema Component Representation

```
<xs:element name="repeat">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref=" add "/>
      <xs:element ref=" component "/>
      <xs:element ref=" delete "/>
      <xs:element ref=" switch "/>
      <xs:element ref=" window "/>
    </xs:choice>
    <xs:attribute name="count" type=" xs:NCName "/>
  </xs:complexType>
</xs:element>
```

Element: **size**

Name	size
Used by (from the same schema document)	Element window , Element condition
Type	xs:string
Nilable	no
Abstract	no

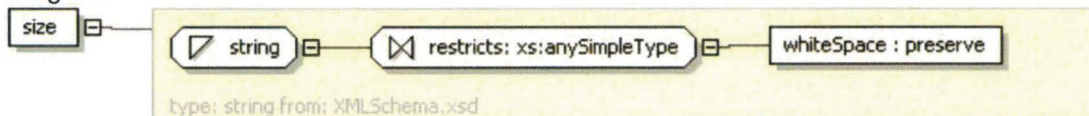
Logical Diagram



XML Instance Representation

```
<size> xs:string </size>
```

Diagram



Schema Component Representation

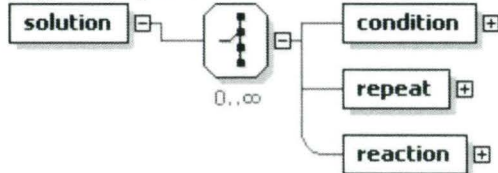
```
<xs:element name="size" type=" xs:string "/>
```

Element: **solution**

Name	solution
-------------	----------

Used by (from the same schema document)	Element pattern
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram

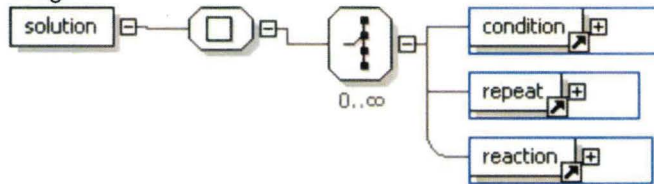


XML Instance Representation

```

<solution>
<!-- Mixed content -->
Start Choice [0..*]
<condition> ... </condition> [1]
<repeat> ... </repeat> [1]
<reaction> ... </reaction> [1]
End Choice
</solution>
    
```

Diagram



Schema Component Representation

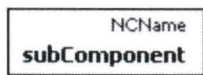
```

<xs:element name="solution">
<xs:complexType mixed="true">
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element ref=" condition "/>
<xs:element ref=" repeat "/>
<xs:element ref=" reaction "/>
</xs:choice>
</xs:complexType>
</xs:element>
    
```

Element: **subComponent**

Name	subComponent
Used by (from the same schema document)	Element delete
Type	xs:NCName
Nilable	no
Abstract	no

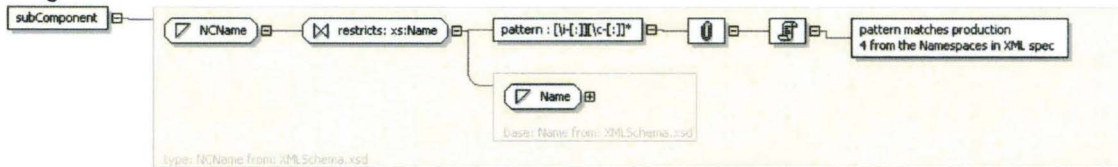
Logical Diagram



XML Instance Representation

```
<subComponent> xs:NCName </subComponent>
```

Diagram



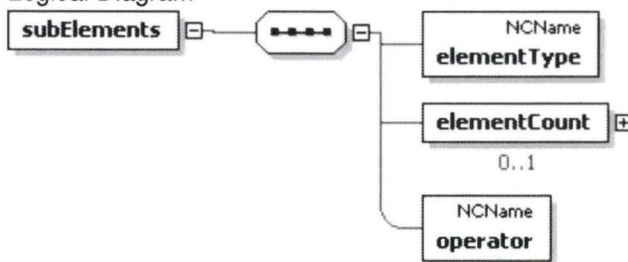
Schema Component Representation

```
<xs:element name="subComponent" type=" xs:NCName "/>
```

Element: subElements

Name	subElements
Used by (from the same schema document)	Element have
Type	Locally-defined complex type
Nilable	no
Abstract	no

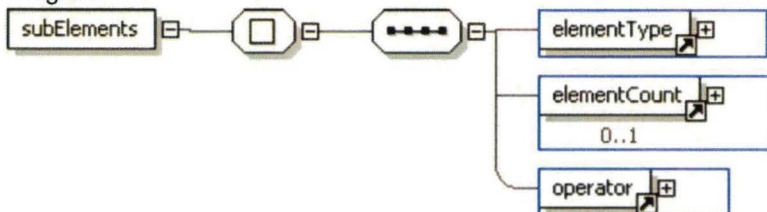
Logical Diagram



XML Instance Representation

```
<subElements>
<elementType> ... </elementType> [1]
<elementCount> ... </elementCount> [0..1]
<operator> ... </operator> [1]
</subElements>
```

Diagram



Schema Component Representation

```
<xs:element name="subElements">
<xs:complexType>
<xs:sequence>
<xs:element ref=" elementType "/>
<xs:element ref=" elementCount " minOccurs="0"/>
```



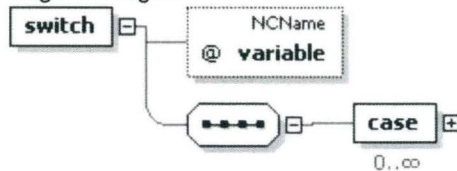
```

<xs:element ref=" operator "/>
</xs:sequence>
</xs:complexType>
</xs:element>

```

Element: **switch**

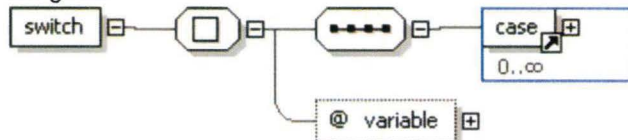
Name	switch
Used by (from the same schema document)	Element repeat
Type	Locally-defined complex type
Nilable	no
Abstract	no

Logical Diagram

XML Instance Representation

```

<switch
variable=" xs:NCName [1]">
<!-- Mixed content -->
<case> ... </case> [0..*]
</switch>

```

Diagram

Schema Component Representation

```

<xs:element name="switch">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref=" case " minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="variable" type=" xs:NCName " use="required"/>
</xs:complexType>
</xs:element>

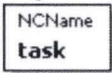
```

Element: **task**

Name	task
Used by (from the same schema document)	Element operation
Type	xs:NCName
Nilable	no

Abstract	no
-----------------	----

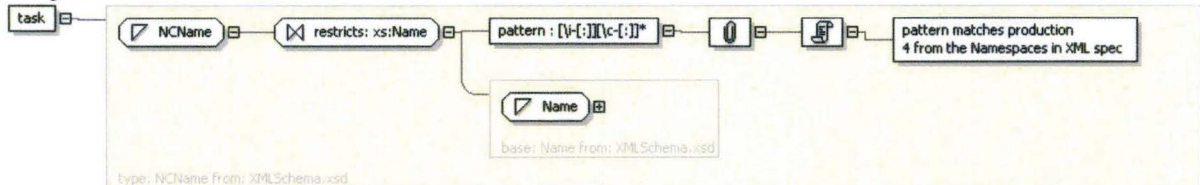
Logical Diagram



XML Instance Representation

```
<task> xs:NCName </task>
```

Diagram



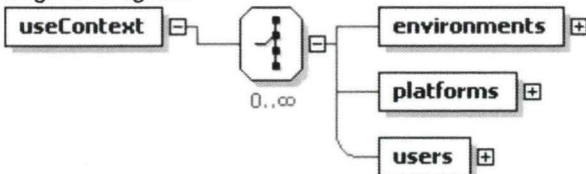
Schema Component Representation

```
<xs:element name="task" type="xs:NCName" />
```

Element: **useContext**

Name	useContext
Used by (from the same schema document)	Element pattern
Type	Locally-defined complex type
Nilable	no
Abstract	no

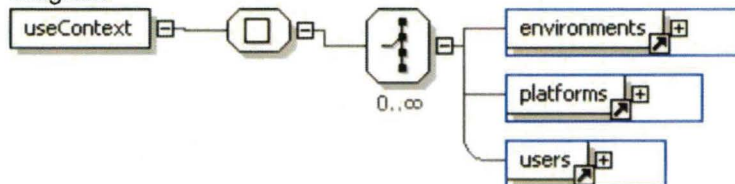
Logical Diagram



XML Instance Representation

```
<useContext>
<!-- Mixed content -->
Start Choice [0..*]
<environments> ... </environments> [1]
<platforms> ... </platforms> [1]
<users> ... </users> [1]
End Choice
</useContext>
```

Diagram



Schema Component Representation

```
<xs:element name="useContext">
<xs:complexType mixed="true">
```

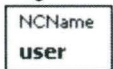
```

<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element ref=" environments "/>
<xs:element ref=" platforms "/>
<xs:element ref=" users "/>
</xs:choice>
</xs:complexType>
</xs:element>
    
```

Element: **user**

Name	user
Used by (from the same schema document)	Element users
Type	xs:NCName
Nilable	no
Abstract	no

Logical Diagram

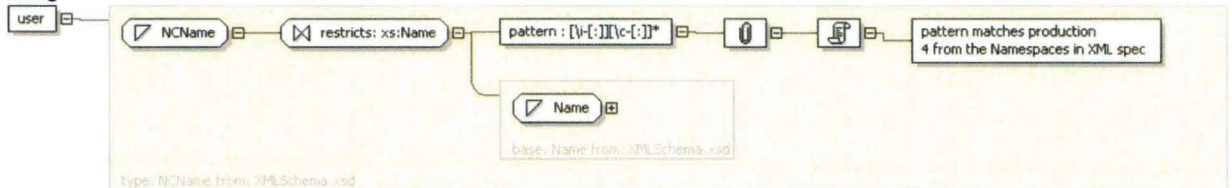


XML Instance Representation

```

<user> xs:NCName </user>
    
```

Diagram



Schema Component Representation

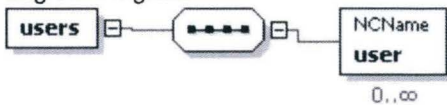
```

<xs:element name="user" type=" xs:NCName "/>
    
```

Element: **users**

Name	users
Used by (from the same schema document)	Element useContext
Type	Locally-defined complex type
Nilable	no
Abstract	no

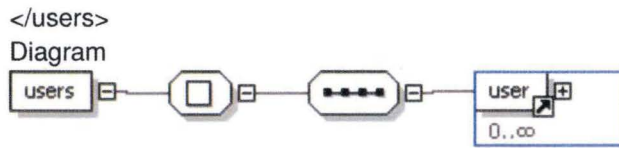
Logical Diagram



XML Instance Representation

```

<users>
<!-- Mixed content -->
<user> ... </user> [0..*]
    
```



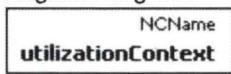
Schema Component Representation

```
<xs:element name="users">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref=" user " minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **utilizationContext**

Name	utilizationContext
Used by (from the same schema document)	Element utilizationContexts
Type	xs:NCName
Nilable	no
Abstract	no

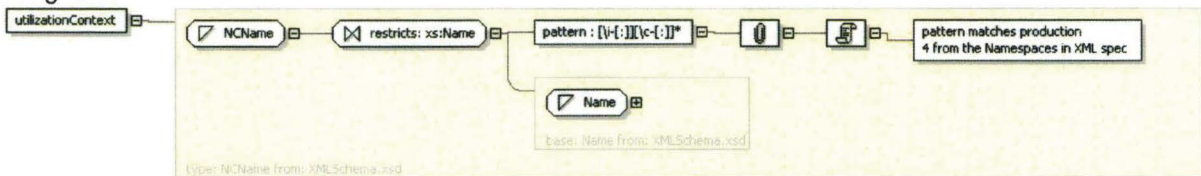
Logical Diagram



XML Instance Representation

```
<utilizationContext> xs:NCName </utilizationContext>
```

Diagram



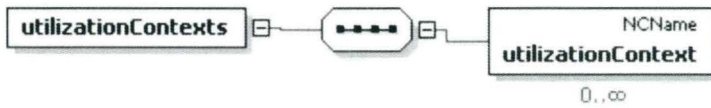
Schema Component Representation

```
<xs:element name="utilizationContext" type=" xs:NCName "/>
```

Element: **utilizationContexts**

Name	utilizationContexts
Used by (from the same schema document)	Element pattern
Type	Locally-defined complex type
Nilable	no
Abstract	no

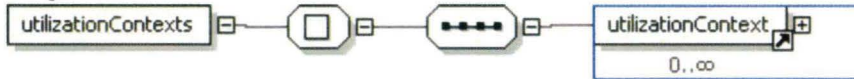
Logical Diagram



XML Instance Representation

```
<utilizationContexts>
<!-- Mixed content -->
<utilizationContext> ... </utilizationContext> [0..*]
</utilizationContexts>
```

Diagram



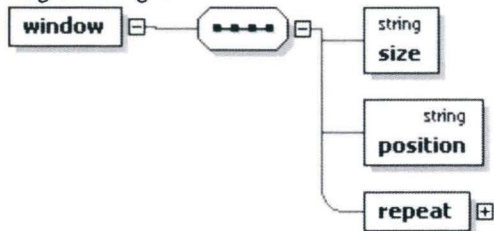
Schema Component Representation

```
<xs:element name="utilizationContexts">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref=" utilizationContext " minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Element: **window**

Name	window
Used by (from the same schema document)	Element repeat
Type	Locally-defined complex type
<u>Nilable</u>	no
<u>Abstract</u>	no

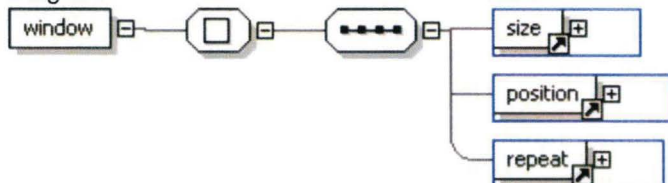
Logical Diagram



XML Instance Representation

```
<window>
<size> ... </size> [1]
<position> ... </position> [1]
<repeat> ... </repeat> [1]
</window>
```

Diagram



Schema Component Representation


```
<xs:element name="window">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element ref="size"/>  
      <xs:element ref="position"/>  
      <xs:element ref="repeat"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```


Annexe C

PaDev : outil de développement de patrons

Dans le cadre la thèse, une maquette d'outil de développement de patrons de conception appelé PaDev a été développée (cf. Figure AC.1). PaDev vise à développer des patrons de conception applicables et utilisables dans notre méthode ; Cet outil s'appuie sur le gabarit proposé dans cette thèse (cf. Chapitre III, §V.2.1). Cette contribution logicielle offre aux développeurs un environnement de développement de patrons avec des fonctions de sauvegarde, restauration, numérotation automatique, ainsi que la possibilité de consulter les patrons existants afin de construire la partie *collaborations* du patron.

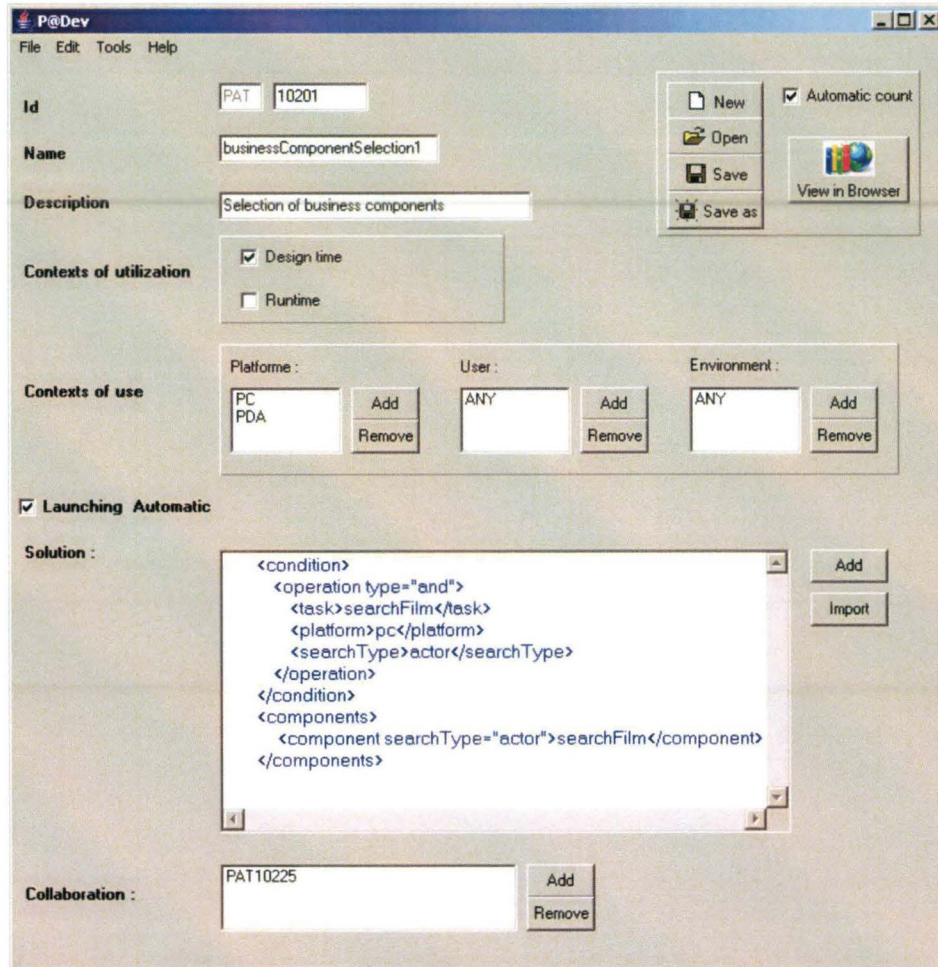


Figure AC.1 : PaDev : outil de développement de patrons de conception

Une liste de balises peut être affichée au développeur (cf. Figure AC.2). Cette liste contient les balises pouvant être utilisées pour décrire une solution (nous avons proposé 35 balises). De plus, le développeur a la possibilité de modifier la liste des balises.

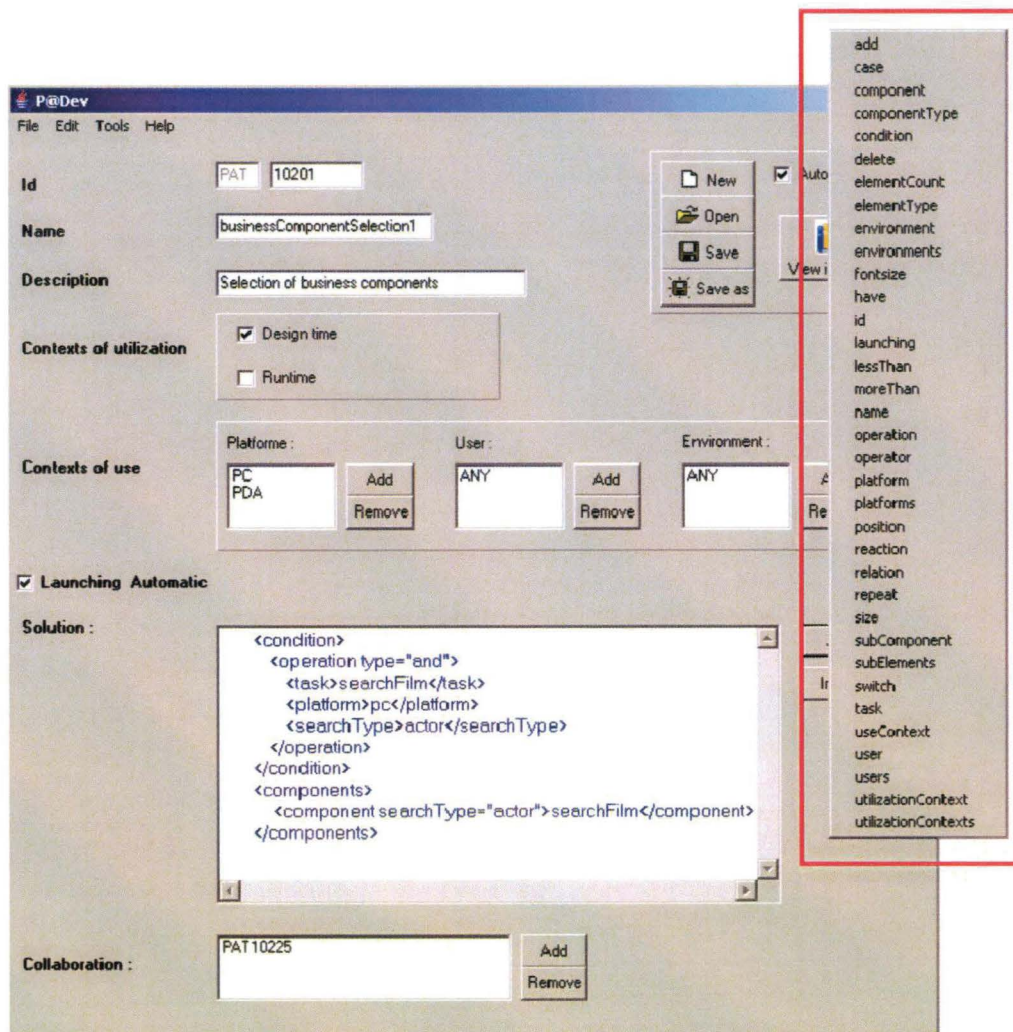


Figure AC.2 : Facilité de description d'une solution en proposant au développeur une liste de balises

PaDev permet également aux développeurs d'intégrer des solutions développées et validées dans un langage de programmation comme par exemple Java (cf. Figure AC.3). PaDev utilise des parseurs XML permettant de représenter un code en XML afin de l'intégrer par la suite dans la partie *solution*.

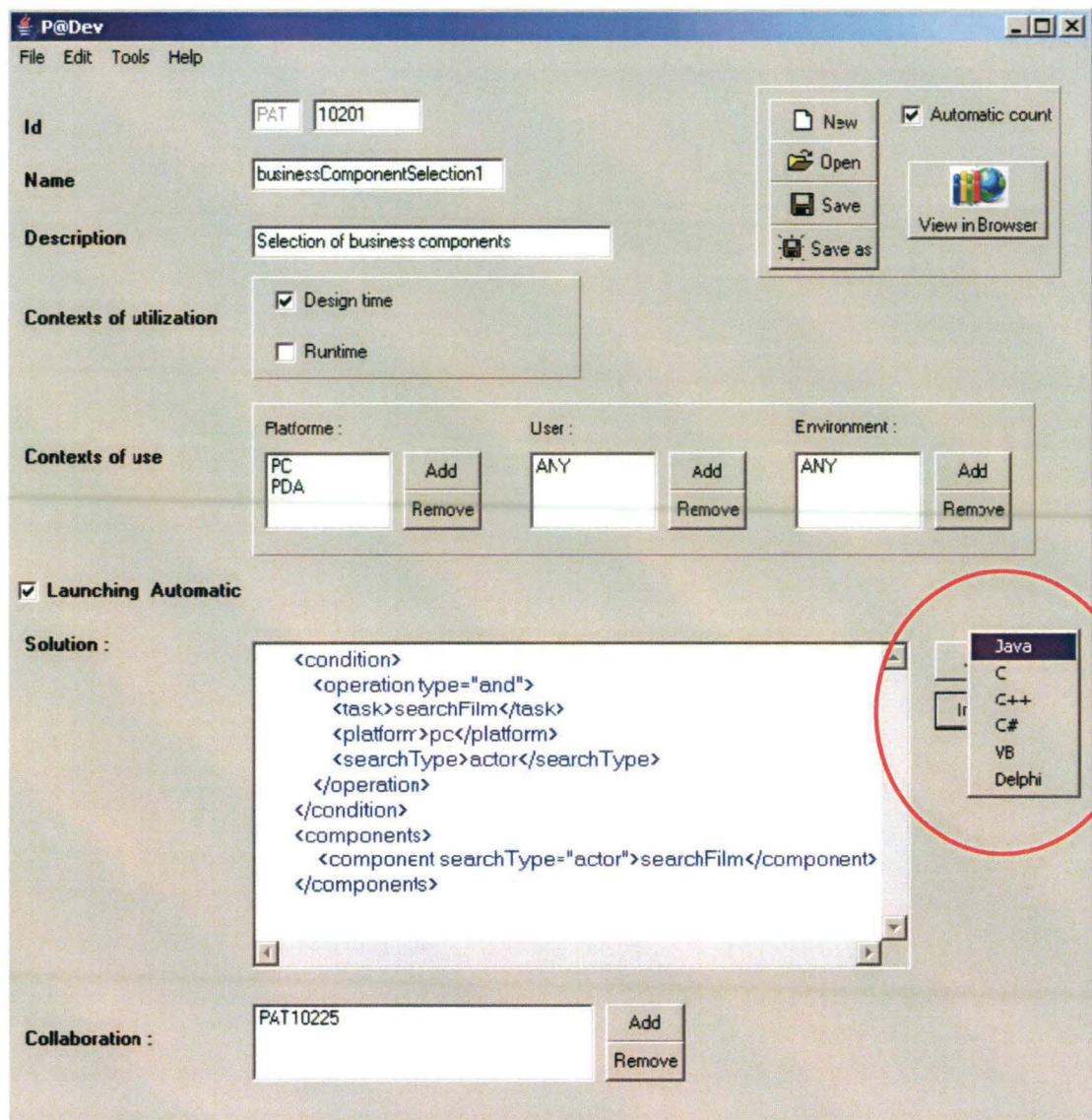


Figure AC.3 : possibilité d'importer une solution développée dans un langage de programmation

Annexe D

Patrons de conception

Nous avons développé huit patrons de conception avec l'outil PaDev. Ces patrons permettent de construire la structure métier du système interactif plastique en s'appuyant sur les règles dans ce but (cf. Chapitre III, §V.5). Le code des patrons de construction de structure métier est présenté ci-dessous :

```

<!-- Premier patron : des composants métier "conteneur" sont associés aux conteneurs abstraits -->
<pattern>// Définir le patron de conception
  <id>PAT00101</id>
  <name>businessStructureBuldingRule1</name>
  <description>Association des composants métier Conteneur aux conteneurs abstraits</description>
  <utilizationContexts>// Contexte d'utilisation
    <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
  </utilizationContexts>
  <useContext>// Contextes d'usage
    <platforms>// sur n'importe quelle plate-forme
      <platform>ANY</platform>
    </platforms>
    <environments>// dans n'importe quel environnement
      <environment>ANY</environment>
    </environments>
    <users>// avec n'importe quel utilisateur
      <user>ANY</user>
    </users>
  </useContext>
  <launching>AUTO</launching>// le patron peut être lancé automatiquement
  <solution>// décrire la solution en commençant avec la condition, ensuite la réaction
    <condition>
      <elementType>abstractContainer</elementType>
    </condition>
    <reaction>
      <add>
        <component>container</component>
      </add>
    </reaction>
  </solution>
  <collaboration>// Liste des patrons collaborant avec le patron
    <patternId>PAT00102</patternId>
    <patternId>PAT00103</patternId>
    <patternId>PAT00104</patternId>
    <patternId>PAT00105</patternId>
    <patternId>PAT00106</patternId>
    <patternId>PAT00107</patternId>
    <patternId>PAT00108</patternId>
  </collaboration>
</pattern>

<!-- Deuxième patron : un composant métier "choix" est associé à un conteneur abstrait qui supervise
l'ensemble des conteneurs abstraits dont les tâches s'exécutent suite au choix effectué -->
<pattern>// Définir le patron de conception
  <id>PAT00102</id>
  <name>businessStructureBuldingRule2</name>
  <description>Ajout des composants métier Choix aux conteneurs abstraits</description>
  <utilizationContexts>// Contexte d'utilisation
    <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
  </utilizationContexts>
  <useContext>// Contextes d'usage
    <platforms>// sur n'importe quelle plate-forme
      <platform>ANY</platform>

```



```

</platforms>
<environments>// dans n'importe quel environnement
  <environment>ANY</environment>
</environments>
<users>// avec n'importe quel utilisateur
  <user>ANY</user>
</users>
</useContext>
<launching>AUTO</launching>// le patron peut être lancé automatiquement
<solution>// décrire la solution en commençant avec la condition, ensuite la réaction
  <condition>
    <operation type="and">
      <elementType>abstractContainer</elementType>
      <have>
        <subElements>
          <elementType>abstractContainer</elementType>
          <elementCount>
            <moreThan>1</moreThan>
          </elementCount>
          <operator>choice</operator>
        </subElements>
      </have>
    </operation>
  </condition>
  <reaction>
    <add>
      <component>choice</component>
    </add>
  </reaction>
</solution>
<collaboration>// Liste des patrons collaborant avec le patron
  <patternId>PAT00101</patternId>
  <patternId>PAT00103</patternId>
  <patternId>PAT00104</patternId>
  <patternId>PAT00105</patternId>
  <patternId>PAT00106</patternId>
  <patternId>PAT00107</patternId>
  <patternId>PAT00108</patternId>
</collaboration>
</pattern>

```

<!-- Troisième patron : des composants métier "panneau" sont associés aux conteneurs abstraits dont les tâches associées s'exécutent en parallèle. Ensuite les composants métier "conteneur" associés aux conteneurs abstraits sont supprimés -->

```

<pattern>// Définir le patron de conception
  <id>PAT00103</id>
  <name>businessStructureBuldingRule3</name>
  <description>Ajout des composants métier Panneau aux conteneurs abstraits</description>
  <utilizationContexts>// Contexte d'utilisation
    <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
  </utilizationContexts>
  <useContext>// Contextes d'usage
    <platforms>// sur n'importe quelle plate-forme
      <platform>ANY</platform>
    </platforms>
    <environments>// dans n'importe quel environnement
      <environment>ANY</environment>
    </environments>
    <users>// avec n'importe quel utilisateur
      <user>ANY</user>

```

```

</users>
</useContext>
<launching>AUTO</launching>// le patron peut être lancé automatiquement
<solution>// décrire la solution en commençant avec la condition, ensuite la réaction
  <condition>
    <operation type="and">
      <elementType>abstractContainer</elementType>
      <have>
        <subElements>
          <elementType>abstractContainer</elementType>
          <elementCount>
            <moreThan>1</moreThan>
          </elementCount>
          <operator>parallel</operator>
        </subElements>
      </have>
    </operation>
  </condition>
  <reaction>
    <repeat count="subElementCount">
      <add>
        <component>panel</component>
      </add>
      <delete>
        <subComponent>container</subComponent>
      </delete>
    </repeat>
  </reaction>
</solution>
<collaboration>// Liste des patrons collaborant avec le patron
  <patternId>PAT00101</patternId>
  <patternId>PAT00102</patternId>
  <patternId>PAT00104</patternId>
  <patternId>PAT00105</patternId>
  <patternId>PAT00106</patternId>
  <patternId>PAT00107</patternId>
  <patternId>PAT00108</patternId>
</collaboration>
</pattern>

```

<!-- Quatrième patron : aucun composant métier ne peut être associé à un conteneur abstrait, si celui-ci supervise un ensemble de sous-conteneurs abstraits ayant déjà des composants métier de type "conteneur", et que leurs tâches associées s'exécutent séquentiellement -->

```

<pattern>// Définir le patron de conception
  <id>PAT00104</id>
  <name>businessStructureBuldingRule4</name>
  <description>Suppression des composants métier Conteneur</description>
  <utilizationContexts>// Contexte d'utilisation
    <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
  </utilizationContexts>
  <useContext>// Contextes d'usage
    <platforms>// sur n'importe quelle plate-forme
      <platform>ANY</platform>
    </platforms>
    <environments>// dans n'importe quel environnement
      <environment>ANY</environment>
    </environments>
    <users>// avec n'importe quel utilisateur
      <user>ANY</user>
    </users>

```

```

</useContext>
<launching>AUTO</launching>// le patron peut être lancé automatiquement
<solution>// décrire la solution en commençant avec la condition, ensuite la réaction
  <condition>
    <operation type="and">
      <elementType>abstractContainer</elementType>
      <have>
        <subElements>
          <elementType>abstractContainer</elementType>
          <operator>sequence</operator>
        </subElements>
      </have>
    </operation>
  </condition><reaction>
    <delete>
      <component>container</component>
    </delete>
  </reaction>
</solution>
<collaboration>// Liste des patrons collaborant avec le patron
  <patternId>PAT00101</patternId>
  <patternId>PAT00102</patternId>
  <patternId>PAT00103</patternId>
  <patternId>PAT00105</patternId>
  <patternId>PAT00106</patternId>
  <patternId>PAT00107</patternId>
  <patternId>PAT00108</patternId>
</collaboration>
</pattern>

```

<!-- Cinquième patron : des composants métier sont associés aux composants abstraits. Les composants métier sont sélectionnés en fonction du type des tâches associées aux composants abstraits -->

```

<pattern>// Définir le patron de conception
  <id>PAT00105</id>
  <name>businessStructureBuildingRule5</name>
  <description>Association des différents composants métier aux composants abstraits</description>
  <utilizationContexts>// Contexte d'utilisation
    <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
  </utilizationContexts>
  <useContext>// Contextes d'usage
    <platforms>// sur n'importe quelle plate-forme
      <platform>ANY</platform>
    </platforms>
    <environments>// dans n'importe quel environnement
      <environment>ANY</environment>
    </environments>
    <users>// avec n'importe quel utilisateur
      <user>ANY</user>
    </users>
  </useContext>
  <launching>AUTO</launching>// le patron peut être lancé automatiquement
  <solution>// décrire la solution en commençant avec la condition, ensuite la réaction
    <condition>
      <operation type="and">
        <elementType>abstractComponent</elementType>
        <task>input</task>
      </operation>
    </condition>
    <reaction>
      <add>

```

```

        <component>inputTxt</component>
    </add>
</reaction>
<condition>
    <operation type="and">
        <elementType>abstractComponent</elementType>
        <task>choice</task>
    </operation>
</condition><reaction>
    <add>
        <component>choice</component>
    </add>
</reaction>...
</solution>
<collaboration>// Liste des patrons collaborant avec le patron
    <patternId>PAT00101</patternId>
    <patternId>PAT00102</patternId>
    <patternId>PAT00103</patternId>
    <patternId>PAT00104</patternId>
    <patternId>PAT00106</patternId>
    <patternId>PAT00107</patternId>
    <patternId>PAT00108</patternId>
</collaboration>
</pattern>

```

<!-- Sixième patron : un composant métier "sortie" est associé aux conteneurs abstraits ayant déjà un composant métier "conteneur" -->

```

<pattern>// Définir le patron de conception
    <id>PAT00106</id>
    <name>businessStructureBuildingRule6</name>
    <description>Ajout des composants métier Sortie</description>
    <utilizationContexts>// Contexte d'utilisation
        <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
    </utilizationContexts>
    <useContext>// Contextes d'usage
        <platforms>// sur n'importe quelle plate-forme
            <platform>ANY</platform>
        </platforms>
        <environments>// dans n'importe quel environnement
            <environment>ANY</environment>
        </environments>
        <users>// avec n'importe quel utilisateur
            <user>ANY</user>
        </users>
    </useContext>
    <launching>AUTO</launching>// le patron peut être lancé automatiquement
    <solution>// décrire la solution en commençant avec la condition, ensuite la réaction
        <condition>
            <operation type="and">
                <elementType>abstractContainer</elementType>
                <have>
                    <component>
                        <componentType>container</componentType>
                    </component>
                </have>
            </operation>
        </condition>
        <reaction>
            <add>
                <component>exit</component>
            </add>
        </reaction>
    </solution>
</pattern>

```



```

    </add>
  </reaction>
</solution>
<collaboration>// Liste des patrons collaborant avec le patron
  <patternId>PAT00101</patternId>
  <patternId>PAT00102</patternId>
  <patternId>PAT00103</patternId>
  <patternId>PAT00104</patternId>
  <patternId>PAT00105</patternId>
  <patternId>PAT00107</patternId>
  <patternId>PAT00108</patternId>
</collaboration>
</pattern>

<!-- Septième patron : un composant métier "validation" est associé aux conteneurs abstraits ayant déjà un
composant métier "conteneur"-->
<pattern>// Définir le patron de conception
  <id>PAT00107</id>
  <name>businessStructureBuildingRule7</name>
  <description>Ajout des composants métier Validation</description>
  <utilizationContexts>// Contexte d'utilisation
    <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
  </utilizationContexts>
  <useContext>// Contextes d'usage
    <platforms>// sur n'importe quelle plate-forme
      <platform>ANY</platform>
    </platforms>
    <environments>// dans n'importe quel environnement
      <environment>ANY</environment>
    </environments>
    <users>// avec n'importe quel utilisateur
      <user>ANY</user>
    </users>
  </useContext>
  <launching>AUTO</launching>// le patron peut être lancé automatiquement
  <solution>// décrire la solution en commençant avec la condition, ensuite la réaction
    <condition>
      <operation type="and">
        <elementType>abstractContainer</elementType>
        <have>
          <component>
            <componentType>container</componentType>
          </component>
        </have>
      </operation>
    </condition>
    <reaction>
      <add>
        <component>validation</component>
      </add>
    </reaction>
  </solution>
  <collaboration>// Liste des patrons collaborant avec le patron
    <patternId>PAT00101</patternId>
    <patternId>PAT00102</patternId>
    <patternId>PAT00103</patternId>
    <patternId>PAT00104</patternId>
    <patternId>PAT00105</patternId>
    <patternId>PAT00106</patternId>
    <patternId>PAT00108</patternId>

```

```

    </collaboration>
  </pattern>

```

<!--Huitième patron : les composants métier sont reliés entre eux en se basant sur la partie <relationships> d'AMXML -->

```

<pattern>// Définir le patron de conception
  <id>PAT00108</id>
  <name>businessStructureBuildingRule8</name>
  <description>Ajout des relations entre les composants métier</description>
  <utilizationContexts>// Contexte d'utilisation
    <utilizationContext>designTime</utilizationContext>// le patron est utilisé à la conception
  </utilizationContexts>
  <useContext>// Contextes d'usage
    <platforms>// sur n'importe quelle plate-forme
      <platform>ANY</platform>
    </platforms>
    <environments>// dans n'importe quel environnement
      <environment>ANY</environment>
    </environments>
    <users>// avec n'importe quel utilisateur
      <user>ANY</user>
    </users>
  </useContext>
  <launching>AUTO</launching>// le patron peut être lancé automatiquement
  <solution>// décrire la solution
    <repeat count="relationshipCount">
      <switch variable="relationType">
        <case relationType="choice">
          <add>
            <relation>choice</relation>
          </add>
        </case>
        <case relationType="exchangeData">
          <add>
            <relation>exchangeData</relation>
          </add>
        </case>
        ...
      </switch>
    </repeat>
  </solution>
  <collaboration>// Liste des patrons collaborant avec le patron
    <patternId>PAT00101</patternId>
    <patternId>PAT00102</patternId>
    <patternId>PAT00103</patternId>
    <patternId>PAT00104</patternId>
    <patternId>PAT00105</patternId>
    <patternId>PAT00106</patternId>
    <patternId>PAT00107</patternId>
  </collaboration>
</pattern>

```

Annexe E

Exemple d'application de l'adaptation vivante

Annexe E : exemple d'application de l'adaptation vivante

Dans le cadre la thèse, nous avons implémenté un exemple d'application de l'adaptation vivante (cf. Figure AC.1). Pendant l'exécution, cette application permet à l'utilisateur final de choisir entre trois espaces d'affichage ; ensuite une adaptation s'effectue en fonction de l'espace d'affichage choisi. L'adaptation est réalisée en utilisant le patron *PAT10201* (cf. Chapitre III, §V.2.1) pour recalculer la taille et la localisation des composants. A tout moment l'utilisateur peut passer d'une plate-forme à une autre (dans la limite des trois plateformes prédéfinies). De plus, le niveau de luminosité peut être défini manuellement (à cause de l'absence de capteurs de luminosité), et une adaptation peut être lancée suite à un changement de luminosité. Enfin l'utilisateur peut changer la taille de police et les couleurs de l'IHM ; ces réactions de l'utilisateur peuvent être prises en compte en cas d'une adaptation à venir.

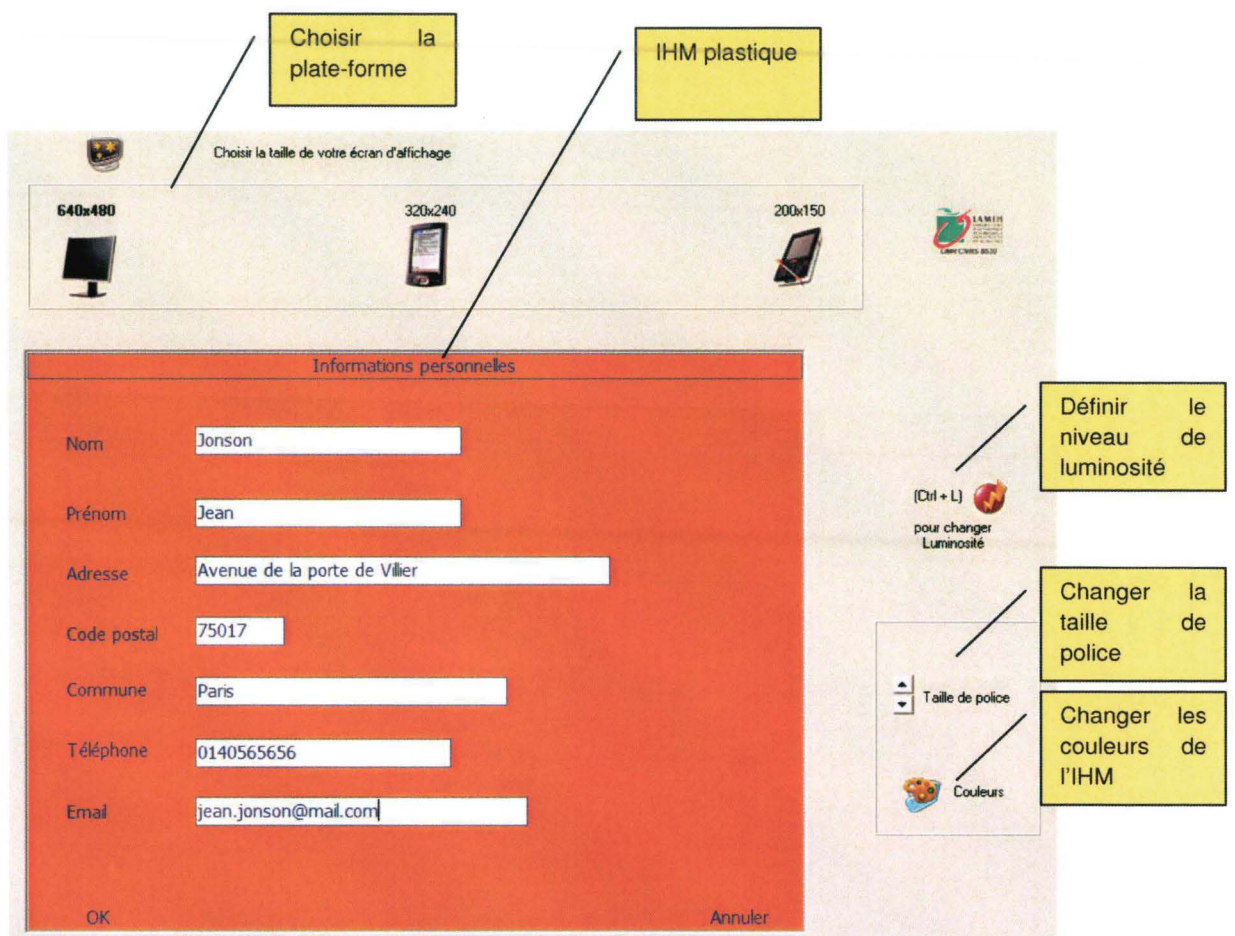


Figure AE.1 : Exemple d'application de l'adaptation vivante



Résumé

De nouvelles perspectives d'utilisation des systèmes interactifs se sont ouvertes grâce à l'évolution à tous les niveaux des moyens de communication, et aux progrès technologiques conduisant en particulier à de nouveaux terminaux de travail mobiles. L'informatique pervasive laisse envisager une nouvelle génération de systèmes interactifs, et nécessite de nouvelles modalités d'interaction homme-machine. Les systèmes interactifs doivent désormais s'adapter à leur contexte d'usage, en préservant leur utilisabilité, sans besoin de re-conception et ré-implémentation coûteuses. Des recherches portent sur de nouveaux types d'IHM, qu'elles soient dites sensibles au contexte, ou encore plastiques avec différentes capacités d'intégration de la notion de plasticité. Cependant, dans la plupart des méthodes, l'adaptation est statiquement prédéfinie par le concepteur à la conception. Ensuite, lors d'une nécessité d'adaptation suite à un changement contextuel, l'IHM doit être renvoyée en phase de conception. De plus, la possibilité d'évaluer la qualité de l'adaptation à l'exécution est souvent absente. Nous nous intéressons dans le cadre de la thèse aux interfaces homme-machine ayant la capacité de s'adapter de manière dynamique à leur contexte d'usage en prenant en compte les changements contextuels sans nécessité de retour à la conception.

Dans ce courant de recherche et en partant du concept d'IHM plastique, notre contribution consiste à générer une telle IHM à partir d'un modèle abstrait d'IHM spécifié déjà dans une méthode de spécification et conception de systèmes interactifs et/ou un modèle de tâche. Notre méthode s'appuie sur la notion de patrons de conception. Ceux-ci sont utilisés au niveau du passage à l'interface concrète et pendant l'adaptation. L'architecture du système s'appuie sur une composition basée sur les composants métier. Ceux-ci ont la capacité de changer dynamiquement leur facette de présentation. Ce principe est adopté en tant que solution pour l'adaptation dynamique au contexte d'usage. Notre méthode s'appuie également sur la notion d'apprentissage. L'intégration d'une technique d'apprentissage permet de continuer à développer la base de connaissance du système à l'exécution afin de préserver l'utilité de l'adaptation. La méthode proposée est illustrée sur deux cas d'étude : la première concerne une application nomade de guidage touristique. La seconde est empruntée au domaine de la supervision industrielle.

Mots clés : Interaction homme-machine, conception, génération, adaptation, contexte, interface plastique

Abstract

Doors to new prospects of using interactive systems have been opened, thanks to the evolution on all the levels of the communication means, and to technological progress leading in particular to new mobile terminal generation. Pervasive computing lets consider a new interactive system generation, and requires new methods of Human-Computer Interaction. From now on the interactive systems must adapt to the context of use, in order to preserve their usability, without need for expensive redesign and reimplementation. Research works relate to new types of User Interface (UI) that are known as context sensitive UI, in other word *plastic*, with various capacities of integration of the *plasticity* notion. However, in the majority of methods, the adaptation is statically predefined by the designer at design stage. Then, following an adaptation necessity because of contextual changes, the UI has to be transferred at design phase. Moreover, the possibility of evaluating the adaptation quality at runtime misses often. We are interested, as part of the thesis, in the UI having the capacity to adapt dynamically to the context of use, by taking into account the contextual changes without any need to return to design phase.

In this research field, and starting from the plastic UI concept, our contribution consists in generating such UI starting from an abstract UI model specified already in an interactive systems specification and design method and/or a task model. Our method is based on the notion of design patterns; those are used during the passage to the concrete UI and during the adaptation. The system architecture is supported by a composition based on business components; those have the capacity to change their presentation facet dynamically. This principle is adopted as a solution for the dynamic adaptation to the context of use. Our method is also based on the notion of machine learning. The integration of a technique of machine learning allows continuing to develop the knowledge base of the system at runtime, in order to preserve the utility of the adaptation. The suggested method is illustrated on two cases of study: the first relates to a nomadic application of touristic guidance. Second is borrowed from the industrial supervision domain.

Keywords: human-mac

xt, plastic user interface

