Adaptation en interaction homme-machine : le cas de la plasticité

David Thevenin

UNIVERSITE JOSEPH FOURIER - GRENOBLE 1
UFR EN INFORMATIQUE ET MATHEMATIQUE APPLIQUEES

N° attribué par la bibliothèque
/__/__/__/__/__/__/__/__/__/__/

T H E S E

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE JOSEPH FOURIER

Discipline : Informatique

présentée et soutenue publiquement

par

David Thevenin

le 21 décembre 2001

Titre :

Adaptation en Interaction Homme-Machine : le cas de la Plasticité

Directeur de thèse :

Joëlle Coutaz

JURY

Mme. Jocelyne Nanard, Rapporteur

M. Jean Vanderdonckt, Rapporteur

M. Jean Caelen, Président

M. Philip Gray, Examinateur

M. Philippe Suignard, Examinateur.

UNIVERSITE JOSEPH FOURIER - GRENOBLE 1
UFR EN INFORMATIQUE ET MATHEMATIQUE APPLIQUEES

T H E S E

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE JOSEPH FOURIER

Discipline : Informatique

présentée et soutenue publiquement

par

David Thevenin

le 21 décembre 2001

Titre :

Adaptation en Interaction Homme-Machine : le cas de la Plasticité

Directeur de thèse :

Joëlle Coutaz

JURY

Mme. Jocelyne Nanard, Rapporteur

M. Jean Vanderdonckt, Rapporteur

M. Jean Caelen, Président

M. Philip Gray, Examinateur

M. Philippe Suignard, Examinateur.

Ce travail de thèse de trois ans n'aurait sûrement pas pu voir le jour sans le concours de nombreuses personnes. Ici je tiens à les remercier pour ce qu'elles m'ont apporté. Jocelyne Nanard, professeur de l'Université de Montpellier II, pour l'intérêt qu'elle a porté à ce travail et l'évaluation qu'elle en a fait ; Philippe Suignard, responsable de projet Recherche Et Développement chez EDF, pour avoir accepté d'être les examinateurs de ce travail.

Un merci également à Samuel, Damien, Nicolas et Noël, quatre étudiants stagiaires. Ils ont participé à l'avancement du projet EDF et à l'application de nos résultats sur PDA et téléphones WAP ;

Avant de commencer par les différents membres du jury de thèse, qui ont jugé mon manuscrit et ma soutenance, je remercierais ... Jean Vanderdonckt, professeur de l'Université Catholique de Louvain, pour son apport scientifique, l'enthousiasme qu'il a toujours eu lors de nos rencontres et sa grande gentillesse ; Je ne vais pas remercie tout ent Ninie, pour la dure tâche de est intéressée problém atique de ion theve nienne de et access oirement ques (:-)) fautes aphe ren contré

Je remercie aussi ma mère, Céline et François pour les relectures ... Enfin, un merci à tous ceux qui ont participé de près ou de loin

Joëlle Coutaz pour m'avoir accepté dans son équipe et pour m'avoir proposé ce sujet de thèse si passionnant. Je la remercie aussi pour les moyens financiers qu'elle matériel s qu'elle a toujours mis à ma disposition durant mes années de DEA et de thèse ; Jean Caelen, Je remercie directeur de Recherche CNRS et Philippe directeur du Laboratoire Gray, “senior CLIPS; lecturer” pour avoir accepté d'être de l'Université président de mon jury de soutenance; Glasgow et en rester là. Je Mais aussi Gaëlle particulièrement Calvary membre s'être attelée à de l'équipe relecture. Elle s'est active IIHM, pour ment à la son travail dans formulation le projet EDF, ses de la phrase idées, et aussi aux quelques d'orthographe pour m'avoir supporté durant les discussions. à la réalisation de cette thèse. Enfin je tiens à dire qu'aujourd'hui, je peux moi aussi l'affirmer sans complexe, le plastique c'est fantastique … alors Joëlle, devenons adeptes du latex …

Table des matières

Table des figures

The Boy with Nails in His Eyes

The Boy with Nails in His Eyes
put up his aluminum tree.
It looked pretty strange
because he couldn't really see.

[Tim Burton 1998]

L'enfant avec des clous dans les yeux

L'enfant avec des clous dans les globes oculaires
monta son arbre en métal,
lequel avait vraiment un drôle d'air
puisque l'enfant n'y voyait que dalle.

Traduit par René Belletto.

Chapitre 1 *Introduction*

Nos travaux de recherche doctorale concernent la plasticité des interfaces homme-machine, une forme particulière d'adaptation au contexte d'utilisation. Cette étude trouve sa justification dans la diversité sans cesse croissante des conditions d'usage et l'absence de cadre de référence et d'outils adaptés à ces nouvelles contraintes.

1. Un constat : diversité des contextes d'utilisation

Depuis ses fondements, le domaine de l'Interaction Homme-Machine (IHM) est marqué par le souci constant de concevoir et de produire des systèmes numériques utiles et utilisables, c'est-à-dire adaptés aux utilisateurs dans leur contexte. La conception contextuelle (*Contextual Design*) de Beyer et Holtzblatt, fondée sur la récolte et l'organisation de données de terrain, permet d'identifier la façon dont les individus travaillent en situation, de détecter les insuffisances et d'améliorer la pratique professionnelle par le biais d'un support numérique. Les modèles de l'Action Située (*Situated Action*) [Schuman 1987], de la Théorie de l'Activité [Bardram 1997] ou encore de la Cognition Distribuée (*Distributed Cognition*)[Halverson 1994] considèrent le contexte d'usage comme pièce maîtresse.

Si l'importance du contexte est acquise en Interaction Homme-Machine, celui-ci n'intervient en pratique qu'en phase amont du processus de développement. Par manque d'outils appropriés, l'adaptation au contexte se dilue progressivement au cours du processus de développement, et le triangle classique « utilisateur-tâche-machine » ne fonctionne que pour un contexte figé a priori, typiquement au bureau pour une station de travail de type PC. Or, la technologie aidant, les sources de variation ne manquent pas avec la diversité des utilisateurs,

des plates-formes d'interaction et de l'environnement dans lequel se tient l'interaction.

1.1. DIVERSITÉ DES UTILISATEURS

Jusqu'ici, un système était conçu pour une classe d'utilisateurs donnée. Avec la pénétration des applications Internet, tous les milieux socio-professionnels sont aujourd'hui concernés. Si la conception d'un site de commerce électronique reste ciblée sur une classe de clients, — par exemple, le système vise les jeunes de 20-25 ans — la concurrence du marché exige néanmoins d'établir une relation personnalisée avec chaque consommateur [Kobsa et al. 2001]. En conséquence, le site doit être capable de s'adapter au visiteur afin de lui offrir les services et produits idoines.

1.2. DIVERSITÉ DES PLATES-FORMES D'INTERACTION

Les dispositifs d'interaction se diversifient par la forme et la finalité. L'ordinateur tout usage se voit prolongé d'appareils dédiés comme les assistants personnels numériques (PDA) et les téléphones mobiles. Inversement, avec la convergence télévision-numérique, un objet à finalité bornée devient un dispositif tout usage. Les PDA, comme des Legos (www.lego.com), incluent progressivement les services de téléphonie et inversement, les fabricants de téléphones font du "portable" un véritable PDA. Avec l'informatique diffuse [Weiser 1993], l'espace et les objets de la vie courante deviennent des entités d'interaction. Il en résulte un foisonnement de solutions techniques correspondant chacune à des usages émergents. Dans cet espace du possible, on observe une constante : l'utilisateur veut avoir le choix. Dès lors, il convient d'envisager pour une application donnée, l'utilisation mixte du gros calculateur comme du petit dispositif, du fixe comme du mobile, de l'ordinateur palpable à l'ordinateur évanescent.

1.3. DIVERSITÉ DES ENVIRONNEMENTS D'INTERACTION

Avec la miniaturisation, l'autonomie des batteries, et les progrès de la communication sans fil et des capteurs de toute sorte, l'accès aux ressources de calcul devient possible en tout lieu et à toute heure. Il en résulte des environnements d'interaction extrêmement diversifiés où les frontières entre milieux familial et professionnel, privé et public, s'estompent.

Voici quelques exemples simples à imaginer ou sur le point d'être disponibles : Le chef de chantier vérifie sur le site, au moyen de son PDA, les mesures du plan enregistrées dans une base de données distante. Des archéologues saisissent leurs observations directement sur le terrain [Renevier et Nigay 2001]. Un PDA, orienté vers un mur, absorbe l'information qui y est projetée. Celle-ci est transformée afin d'en conserver la lisibilité sur le petit écran. Le touriste passant à proximité d'un site historique est informé d'événements sensés l'intéresser [Cheverst et al. 2001]. Chez soi, le PDA sert de télécommande universelle. Il s'adapte automatiquement à l'objet le plus proche, mais ne fonctionne pas lorsqu'il est actionné par un jeune enfant. En réunion, la

sonnerie du téléphone passe automatiquement en mode vibreur ou au contraire utilise une sonnerie discernable dans un milieu bruyant. A la gare, le système de réservation de titre de transport conduit efficacement l'acheteur à son but et sans erreur malgré les conditions de stress (plusieurs personnes attendent et le train part dans cinq minutes!). Mais au domicile, le système se permet quelques digressions, offrant la possibilité de consulter les voyages en promotion.

En résumé, le contexte d'utilisation d'un système change et, avec Internet, la diversité des utilisateurs grandit. On assiste progressivement à la création d'un tissu informationnel numérique extrêmement malléable qui appelle de nouveaux requis en termes de processus et d'outils de développement

2. Nouveaux requis : processus et outils pour le développement d'IHM multicibles

Les IHM doivent être conçues pour intégrer les nouvelles opportunités d'adaptation au changement. De monocible (une classe d'utilisateurs donnée œuvrant dans un contexte d'utilisation fixé par avance), les IHM doivent devenir multicibles.

Définition. Une **cible** se définit par le triplet <classe d'utilisateurs, plate-forme, environnement >

où:

- "plate-forme" désigne le support matériel et logiciel qui sous-tend l'interaction. La taille de l'écran, les dispositifs d'interaction, les capacités de calcul et de communication caractérisent une plate-forme d'interaction tels un PC, un PDA ou un téléphone portable.
- "environnement" dénote le milieu dans lequel s'exerce l'interaction homme-machine. Il comprend un ensemble d'informations, périphériques à la tâche en cours, mais susceptibles de l'influencer [Rey 2001] : le lieu géographique ou symbolique (à domicile, au cinéma, dans la rue, dans le train), le jour (ouvrable, férié), l'heure (de jour, de nuit), l'ambiance (fréquenté, bruyant).

Définition. Une **IHM multiplate-forme** est une IHM multi-cible pour laquelle seul l'élément "plate-forme" est susceptible de changer. La conception multiplate-forme est un sous-ensemble de la conception multicible, visant seulement la production d'interfaces pour différentes plates-formes d'interaction.

La conception et le développement d'IHM multicible présentent des difficultés majeures de génie logiciel :

- Le développement parallèle de plusieurs IHM cibles pose des problèmes de partage de ressources et de communication entre les différents projets (cohérence des spécifications, réutilisation de code, etc.).
- Il en va de même pour la maintenance qui doit assurer une évolution simultanée des mises à jour et des corrections de bogues pour chaque cible.
- Plus spécifiquement, le développement multiplate-forme nécessite des connaissances techniques sur chaque plate-forme ciblée (PalmPilot, PocketPC, Psion, etc.)

Au-delà des problèmes de conduite de projet, le multicible explose sous l'effet de la combinatoire <utilisateur, plate-forme, environnement>. Par exemple, on n'imagine pas développer et maintenir manuellement chaque page Web d'un site pour chaque plate-forme (1024x768, 800x600, 640x480, 320x240, HTML, WML, etc.) ! De plus, la cible peut varier dynamiquement à l'exécution. Par exemple, la nature des articles susceptibles d'intéresser un nouveau consommateur ne peut être décidée à la conception. Elle doit être calculée, une fois le comportement du consommateur connu. Il en va de même pour l'identification de l'environnement d'interaction : chez soi, au bureau, en un lieu bruité, public ou privé, etc.

Au bilan, il convient de proposer des outils qui masquent la complexité de développement et de maintenance d'IHM multicibles. Ces outils doivent permettre la production de composants logiciels qui, connectés à des mécanismes généraux d'exécution, permettent l'adaptation dynamique d'IHM. Ces requis appellent les objectifs de nos travaux de recherche pour le cas particulier des IHM plastiques.

3. Objectifs : outils pour la plasticité des IHM

Le problème des IHM multicibles est, à notre sens, trop complexe pour être envisagé dans toute sa généralité. En particulier, l'adaptation à l'utilisateur, traitée sans beaucoup de succès depuis une vingtaine d'année, est difficile à cerner. Aussi, nous choisissons de restreindre nos objectifs d'adaptation au cas de la plasticité.

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

Par contraintes matérielles, nous entendons les ressources physiques nécessaires à une exécution nominale de l'IHM. Surface d'affichage, processeur et réseaux en sont des exemples. Une interface plastique s'exécute sur station de travail, assistant personnel ou téléphone portable sans dégradation de son utilisabilité : elle offre une accessibilité matérielle. L'adaptation à l'environnement relève de l'accessibilité topologique et temporelle du dispositif d'interaction. L'utilisabilité est évaluée sur la base de propriétés [Coutaz et Nigay 2001] (facteurs et critères) énoncées dans le cahier des charges pour lesquelles on aura précisé le domaine de valeurs acceptables pour les différentes conditions d'usage matérielles et environnementales.

En plasticité, l'adaptation est envisagée pour le bien-être d'une classe stéréotype d'utilisateurs spécifiée dans le cahier des charges, mais elle ne tient pas compte des changements intra- et inter- personnels (mental, physique, etc.). Autrement dit, une IHM plastique est un cas particulier d'IHM multicible : dans le triplet <classe d'utilisateurs, plateforme, environnement>, la classe d'utilisateurs est fixée a priori sous forme de stéréotype.

Définition. On nomme **contexte d'interaction** le doublet <plateforme, environnement>. Ainsi, étant donné une classe d'utilisateurs et une application, une interface plastique se remodèle à façon afin de s'accommoder aux variations du contexte d'interaction.

On notera que, par définition,

- **IHM plastique** est synonyme d'**IHM multicontexte** à condition que cette IHM maintienne, pour tout contexte d'interaction, le niveau d'utilisabilité fixé par avance,
- Notre notion de contexte d'interaction est résolument ancrée dans le monde physique.

Ayant précisé la couverture des problèmes d'adaptation auxquels nous nous intéressons, notre objectif est le suivant : Définir et offrir un cadre logiciel général d'aide au processus de conception et de mise en œuvre d'IHM plastiques. Pour atteindre ces objectifs, nous proposons une approche par modèle.

4. Approche par modèle

L'analyse de l'état de l'art en matière d'adaptation révèle des outils parcellaires de bas niveau d'abstraction. Il s'agit de :

- boîtes à outils couvrant tout juste la portabilité de code (comme

Java/Swing),

- concepts à peine naissants comme les contexteurs [Rey 2001] et les context widgets [Dey 2000] pour la capture et la modélisation de l'environnement,
- mécanismes à base de règles ou de variables actives pour la détection de changement de situation.

Si ces outils servent de composants logiciels, utiles pour l'adaptation à l'exécution, ils ne répondent pas au requis de généricité nécessaire au processus amont de conception et de développement.

Nous pensons que l'approche par spécification est une voie naturelle au processus de développement à condition de veiller au rapport coût/bénéfice en matière de spécifications. Nous proposons une approche orientée modèle de type MB-IDE (*Model-Based Interface Development Environmen*t) qui a fait partiellement ses preuves dans le passé dans le milieu industriel, mais qui, au vu de l'évolution des requis, mérite d'être reconsidérée.

Nous proposons un processus de référence outillé qui :

- structure les étapes de production d'IHM plastiques en niveaux d'abstraction parfaitement identifiés jusqu'à l'obtention d'IHM plastiques exécutables,
- intègre des principes du génie logiciel pour la gestion des configurations et des versions,
- minimise et contrôle les dépendances entre les différents produits du processus,
- permet de capitaliser les spécifications réduisant ainsi les coûts souvent rédhibitoires de spécification,
- respecte les méthodes de conception et de développement des développeurs en leur permettant d'intervenir librement à n'importe quel niveau d'abstraction du processus et, si besoin, de surcharger les produits obtenus par automatisation,
- permet d'envisager la rétro-conception d'IHM existantes en vue de les rendre plastiques.

Ayant défini un cadre structurant général pour la production d'IHM multicible, nous validons nos préconisations conceptuelles par la réalisation logicielle de deux démonstrateurs aux qualités complémentaires : ARTStudio et MMS.

- ARTStudio (*Adaptation through Reification and Translation Studio*) est un générateur d'interface plastique construit selon les principes de notre cadre de référence. Il vise une conception assistée : il inclut des générateurs, mais aussi des éditeurs graphiques permettant au concepteur d'intervenir à tous les niveaux du processus de produc-

tion d'IHM plastiques.

- MMS vise une plasticité dynamique. Il montre la réalisation d'une architecture logicielle et d'un mécanisme de médiateur permettant l'adaptation à l'exécution. Il intègre des règles d'adaptation plus poussées que celles d'ARTStudio.

5. Structure du mémoire

La structure du mémoire suit les points principaux de notre méthode :

Au chapitre 2 suivant, nous analysons les taxonomies et les outils existants en rapport avec l'adaptation. De cette étude de l'état de l'art, nous proposons notre propre espace de classification : la **fleur des outils pour IHM multicibles** qui met en évidence les lacunes des outils existants. Ce constat nous amène à de nouvelles propositions en matière d'outils de spécification d'IHM multicibles.

Au chapitre 3, nous énonçons un ensemble de principes qui nous semblent fondamentaux en spécification d'IHM multicible. Il s'agit des principes de **factorisation** et de **décoration** qui militent en faveur de la capitalisation de connaissances. Factorisation et décoration s'inscrivent dans un cadre théorique qui, fondé sur la coopération des principes de **réification** et de **traduction**, constitue un **processus de référence générique et conceptuel** pour la définition d'outils de production d'IHM multicible.

Au chapitre 4, nous instancions le processus de référence pour le cas de la génération d'IHM plastiques. Nous proposons le **processus en fermeture éclair** avec les modèles que ce processus implique. Parmi ces modèles, nous proposons l'Interface Utilisateur Abstraite, l'Interface Utilisateur Concrète et l'Interface Utilisateur Finale qui correspondent chacun à un niveau de réification du processus de génération.

Les chapitres 5 et 6 finissent de concrétiser le processus de référence et ses modèles avec la mise en œuvre qui en a été faite dans ARTStudio. Le chapitre 5 présente et justifie le choix des formalismes et notations employés pour la spécification et la production des modèles : UML, XML, ConcurTaskTree, et Java. Le chapitre 6 décrit les difficiles problèmes de l'inférence en génération d'IHM et comment ils ont été résolus dans ARTStudio.

Une conclusion résume notre contribution, énonce les points forts et limites de ce travail qui, à leur tour, justifient de nouvelles perspectives.

Trois annexes et un glossaire complètent le mémoire :

- en annexe 1, un rappel des propriétés auxquelles nous faisons référence dans le mémoire (adaptabilité, adaptativité, etc.) ;
- en annexe 2, le détail d'implémentation sur ARTStudio, cœur applicatif de notre travail ;
- en annexe 3, le démonstrateur, MMS, qui nous a permis de tester des points sensibles sur l'adaptation dynamique ;
- en fin de mémoire, un glossaire terminologique.

Partie 1 Adaptation et Plasticité

Chapitre 2 *De l'adaptation des IHM : analyse et taxonomies*

L'étude de l'adaptation des systèmes numériques n'est pas un phénomène nouveau. L'Intelligence Artificielle (IA), en quête de fabriquer des machines intelligentes, s'est la première intéressée à l'automatisation de l'adaptation. En ingénierie logicielle classique, on se contentait jusqu'ici d'identifier les paramètres de personnalisation sur lesquels il était possible de jouer manuellement. Les menus de préférences en sont une illustration classique. Si l'IA, par le tout automatique, visait des systèmes adaptatifs, l'ingénierie logicielle se concentrait sur le caractère adaptable du système. Dans le premier cas, le système modifie son comportement sans intervention explicite de l'utilisateur. On dit que le système est doué d'adaptativité. Dans le second cas, le système est adapté sur intervention explicite et volontaire de l'utilisateur. Le système présente alors des facultés d'adaptabilité [Coutaz et Nigay 2001].

Mais l'adaptativité comme l'adaptabilité ne se manifestent pas seulement à l'exécution. Ces propriétés sont le résultat d'un processus de conception dirigé par les questions quintiliennes "qui, quoi, pourquoi, comment, quand", autrement dit, pour le sujet qui nous concerne : qui s'adapte ou qui adapte ? à quoi s'adapter ? pourquoi s'adapter ? comment et quand s'adapter ? etc. Ces questions récurrentes sont à l'origine de plusieurs taxonomies qui chacune offre un point de vue sur l'adaptation. A notre tour, nous proposons une perspective complémentaire centrée sur l'ingénierie logicielle par le biais de la décomposition fonctionnelle du modèle Arch. Ce nouveau point de vue sur l'adaptation permet d'analyser les outils actuellement disponibles en matière d'adaptation du point de vue de leur couverture fonctionnelle. L'analyse des lacunes justifie nos propres propositions de cadre conceptuel et d'outils présentés dans les chapitres suivants du mémoire.

Ce chapitre est structuré comme suit :

- La section 1 présente les taxonomies les plus significatives de la lit-

térature ;

- La section 2 présente notre point de vue système de l'adaptation ;
- La section 3 présente notre propre classification en matière d'IHM multicibles et d'outils permettant de produire ces IHM ;
- Les sections 4 et 5 présentent notre analyse des outils actuellement disponibles selon les éléments de notre classification ;
- Enfin dans la dernière section, nous rappelons les manques des outils existants.

1. Taxonomies ou points de vue sur l'adaptation

Les années quatre-vingt ont été riches en recherche sur l'adaptation donnant lieu à de nombreuses publications au début des années quatre-vingt-dix. Nous commençons par celles-ci et notamment par les travaux menés dans le projet AID [Browne et al. 1990a] et les propositions répertoriées dans l'ouvrage [Schneider-Hufschmidt et al. 1993]. Ces résultats sont motivés les uns et les autres par l'adaptation au comportement de l'utilisateur. Puis nous présentons des taxonomies plus récentes, souvent moins riches, mais sensibles au problème des IHM multicibles. Ces différents travaux montrent l'étendue du problème de l'adaptation.

1.1. TAXONOMIES CENTRÉES SUR L'UTILISATEUR

Les premiers travaux dans le domaine de l'adaptation en interface homme-machine sont centrés sur une adaptation à l'utilisateur. Dans cette section nous allons voir les principaux résultats issus notamment de deux ouvrages, que nous considérons comme des références [Browne et al. 1990a] et [Schneider-Hufschmidt et al. 1993].

AID

Le projet AID (*Adaptive Intelligent Dialogue*) traite de la conception et de la mise en œuvre de systèmes capables de s'adapter à l'hétérogénéité des utilisateurs [Browne et al. 1990a]. Dans cette optique, une interface adaptative se définit comme une interface capable de changer son comportement pour correspondre à un individu ou à un groupe d'individus. Notons qu'en l'occurrence, l'adaptation se fait automatiquement et que seul l'utilisateur final (décrit par ses motivations, ses envies, ses facultés psychomotrices, d'apprentissage et de compréhension, etc.) est le motif d'adaptation.

Dans AID, quatre composants sont susceptibles de s'adapter à l'utilisateur : l'aide (en accord avec le niveau de compréhension de l'utilisateur), les messages d'erreur (comme source de réconfort et de mise en confiance), le langage de commande (dont la terminologie doit être conforme à celle de l'utilisateur) et le style de dialogue. Nous consta-

tons que les formes d'adaptativité envisagées dans AID, parmi les tout premiers travaux sur l'adaptation, collent aux recommandations ergonomiques de base de l'époque et n'envisagent pas l'adaptation au contexte d'interaction (plate-forme et environnement).

Si, dans AID, la couverture fonctionnelle de l'adaptativité paraît rudimentaire, la couverture méthodologique est néanmoins significative. En particulier, [Browne et al. 1990b] proposent six classes de métriques servant de guide de pensée au processus de conception, de mise en œuvre et d'évaluation d'IHM adaptative : les classes "objectif", "évaluation de théorie", "déclencheur", "recommandation", "généralité" et "implémentation".

- **Objectif**. Cette classe de métriques spécifie les motivations de l'adaptation. Elle répond à la question du "pourquoi?" Par exemple, l'adaptation doit permettre d'améliorer la vitesse d'interaction, de diminuer le taux d'erreurs, de réduire le temps d'apprentissage, d'améliorer la satisfaction de l'utilisateur, etc.
- **Evaluation de théorie**. Cette classe de métriques est nécessaire lorsque les moyens pour atteindre l'objectif s'appuient sur une théorie non testée. Par exemple, l'adaptation pourrait s'appuyer sur une théorie qui vise à réduire le taux d'erreurs pour améliorer la satisfaction de l'utilisateur. Cette hypothèse, non testée, doit être vérifiée et, par conséquent, doit donner lieu à une collecte de données à l'exécution. Le taux d'erreur, dans ce cas, constitue une mesure d'évaluation de la théorie, mesure qui peut être utilisée par le système ou le concepteur pour changer de stratégie d'adaptation ou de théorie.
- **Déclencheur**. Cette classe de métriques dénote les aspects quantifiés de l'interaction, causes de l'adaptation. Ces déclencheurs peuvent être la latence du système ou de l'utilisateur, un taux ou un type d'erreurs, une suite d'actions utilisateur, etc. Les déclencheurs déterminent la nature des données à collecter et à interpréter à l'exécution. Ils constituent la source du mécanisme d'adaptation et/ou l'entrée d'une théorie.
- **Recommandation**. Les métriques de recommandation décrivent les actions d'adaptation, c'est-à-dire les réactions, à appliquer. Par exemple, fournir davantage d'aide ou augmenter la taille des cibles si le système venait à constater un taux d'erreur élevé pour les tâches de sélection par pointage. Ce type de métriques décrit les sorties d'une théorie.
- **Généralité**. Les métriques de généralité précisent l'applicabilité, c'est-à-dire les limites de l'adaptativité du système en termes, notamment, de population d'utilisateurs ou de domaine de tâches. On constate que le contexte d'interaction n'est pas évoqué.
- **Implémentation**. Cette classe de métriques permet d'évaluer les effets des mécanismes d'adaptation sur les performances générales

du système.

Ces différentes classes de métriques, on le voit, suscitent les bonnes questions dès la conception avec leurs conséquences sur les mécanismes à mettre en œuvre. Totterdell et Rautenbach, également membres du projet AID, s'appuient sur la nature de ces mécanismes d'exécution pour classer les systèmes adaptatifs en six niveaux de sophistication croissante [Totterdell et Rautenbach 90] :

- **Système câblé** : (designed system) le comportement du système est défini à la conception. Aucune adaptation possible.
- **Système adaptable** : (adaptable system) le système est personnalisable sur intervention explicite de l'utilisateur qui peut agir sur les paramètres fixés par le concepteur.
- **Système adaptatif** : (adaptive system) le système est doué de discrimination. Il sait reconnaître la situation (parmi plusieurs déclencheurs fixés par le concepteur) et adopte la réaction (recommandation) fixée elle aussi par le concepteur pour cette situation. Mais le système est incapable d'évaluer l'effet de sa réaction.
- **Système autorégulateur** : (self-regulating adaptive system) le système est doué de facultés d'auto-évaluation. Non seulement il reconnaît la situation parmi les déclencheurs prévus par le concepteur, mais utilise, pour effectuer son choix de réaction, une fonction de *feedback* sur lui-même par essai-erreur. Autrement dit, il utilise une métrique qui évalue l'effet de la réaction retenue précédemment pour ce même déclencheur.
- **Système automédiateur** : (self-mediating system) Le système a les caractéristiques de l'autorégulation, mais en plus, il est capable de résoudre le problème de l'adaptation par planification et peut évaluer a priori l'effet d'une réaction (alors qu'un système autorégulateur n'évalue l'effet qu'a posteriori).
- **Système automodificateur** : (self-modifying adaptive system) La phase ultime du système adaptatif autonome. Il s'agit d'un système doué de généralisation qui, par ses méta-connaissances, a la capacité d'apprendre de nouveaux déclencheurs et de nouvelles réactions.

La table 1 caractérise les six niveaux de systèmes adaptatifs selon qui, du système, du concepteur, ou de l'utilisateur, décide de la nature des déclencheurs, du choix de la réaction et de l'évaluation de l'effet de la réaction.

Table1. Taxonomie des systèmes adaptatifs (d'après [Totterdell et Rautenbach 90], pp. 77.

Niveau d'adaptativité	Identification des déclencheurs	Choix des réactions (recommandations)	Evaluation des réaction
Système câblé	Concepteur	Concepteur	Concepteur
Système adaptable	Concepteur	Utilisateur	Concepteur
Système adaptatif	Concepteur	Système	Concepteur

Table1. Taxonomie des systèmes adaptatifs (d'après [Totterdell et Rautenbach 90], pp. 77.

Niveau d'adaptativité	Identification des déclencheurs	Choix des réactions (recommandations)	Evaluation des réaction
Système autorégulateur	Concepteur	Système	Système
Système automédiateur	Concepteur	Système	Système
Système automodificateur	Système	Système	Système

La taxonomie du projet AID met essentiellement l'accent sur les niveaux d'autonomie des mécanismes d'adaptation. Autrement dit, le "quand" de l'adaptation est résolument orienté vers l'exécution. Toutefois, la distinction explicitée dans la table 1 sur les prises de décision entre système, utilisateur et concepteur, exprime de manière implicite la distinction entre les étapes de conception et d'exécution. La taxonomie de Dieterich et al. présentée maintenant répond de manière plus complète à nos critiques.

Taxonomie de Dieterich et al.

Dans un ouvrage sur les interfaces adaptatives [Schneider-Hufschmidt et al. 1993], Dieterich et al. proposent un espace de conception multidimensionnel qui permet de situer un ensemble représentatif de systèmes adaptatifs [Dieterich et al. 1993]. Si nous retrouvons des dimensions similaires aux métriques de [Browne et al. 1990b], cette taxonomie introduit de nouveaux axes comme les étapes de l'adaptation. Mais comme dans le projet AID, l'adaptation à l'exécution reste le centre de l'analyse : selon Dieterich et al., seule une adaptation dynamique peut correspondre aux besoins spécifiques de l'utilisateur final. Reprenons en détail chacune des dimensions de leur taxonomie en relevant les éléments communs et les différences avec les propositions taxonomiques du projet AID.

Les dimensions communes avec AID. Il s'agit des objectifs de l'adaptation, des déclencheurs que Dieterich nomme "information considérée pour l'adaptation" et des recommandations assimilables chez Dieterich aux "constituants adaptés".

Les autres dimensions de l'espace sont nouvelles, explicitent ou affinent les propositions du projet AID.

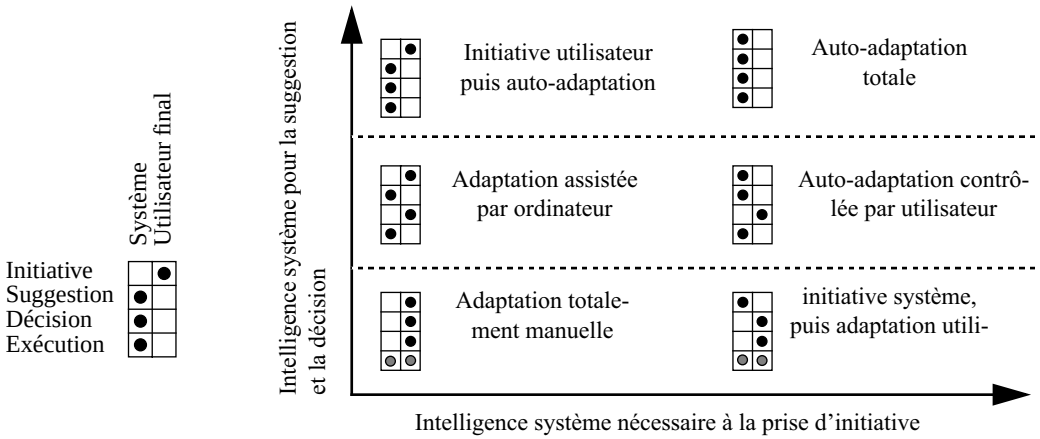
Acteurs et étapes du processus d'adaptation. [Dieterich et al. 1993] identifient cinq intervenants possibles dans le processus d'adaptation : le concepteur, l'administrateur du système, l'expert local, l'utilisateur final et le système lui-même. Estimant que les adaptations réalisables par les acteurs de type concepteur, administrateur ou expert local, ne peuvent convenir qu'à un groupe d'utilisateurs (donc pas à l'individu), Dieterich et al. ne retiennent, comme dans le projet AID, que les acteurs "utilisateur" et "système".

De manière orthogonale aux acteurs, [Dieterich et al. 1993] distinguent, dans le processus d'adaptation, quatre étapes successives : initiative, suggestion, décision et exécution.

- L'initiative représente la décision d'un acteur de suggérer une adaptation ;
- La suggestion représente les propositions de réaction ou, selon la terminologie AID, les recommandations ;
- La décision correspond au processus de choix parmi les propositions de réaction ;
- L'exécution est la mise en œuvre de la réaction choisie.

A partir des deux acteurs (système et utilisateur) et des quatre étapes (initiative, suggestion, décision, exécution), on obtient 16 combinaisons possibles de répartition des responsabilités dans le processus d'adaptation. La figure 1 montre six combinaisons intéressantes définissant chacune une classe d'adaptation. Aux deux extrêmes, le sys-

Figure 1. Taxonomie de [Dieterich et al. 1993]. Correspondance entre acteurs et étapes d'adaptation. Chaque ligne d'une matrice représentant une étape, un • dans une case de cette ligne indique qui, du système ou de l'utilisateur, a la responsabilité de l'étape.



tème, en haut à droite, est totalement auto-adaptatif puisqu'il prend en charge toutes les étapes du processus ; et en bas à gauche, le système est adaptable sur intervention explicite de l'utilisateur sur toutes les étapes. Les trois matrices formant la colonne de gauche dénotent tous les cas où la prise d'initiative revient à l'utilisateur tandis que dans les matrices de la colonne de droite, le système prend l'initiative. L'intelligence du système en matière d'initiative croît donc sur un axe horizontal de gauche à droite. Sur l'axe vertical, l'intelligence du système croît en matière de suggestion et de prise de décision : les matrices de la ligne du bas correspondent à des systèmes où les suggestion et décision reviennent à l'utilisateur tandis qu'elles reviennent au système sur la ligne du haut. La ligne du centre est intéressante : le système propose, l'utilisateur dispose.

Dans AID, Totterdell et Rautenbach insistent sur l'évaluation de l'effet de l'adaptation qui revient, selon le cas, au concepteur ou au système. Chez Dieterich et al. cette étape n'est pas explicite. Elle devrait à notre sens constituer une cinquième étape du processus d'adaptation puisqu'elle permet d'informer les futures adaptations.

Moment de l'adaptation. [Dieterich et al. 1993] identifient trois moments possibles pour l'adaptation d'une interface : avant la première utilisation, pendant l'utilisation et entre deux utilisations.

- Une adaptation avant la première utilisation est une adaptation faite par le concepteur, l'expert local ou l'administrateur. Elle correspond notamment à une configuration du logiciel à l'installation.
- Une adaptation en cours d'exécution (considérée comme la plus intéressante) se passe de manière continue pendant l'utilisation du logiciel par l'utilisateur final.
- Une adaptation entre deux utilisations correspond à une adaptation qui demande de relancer l'application ou une reconfiguration par l'expert local, l'administrateur ou l'utilisateur.

Constituants adaptés. Alors que dans AID, Browne et al. laissent libre la nature de l'adaptation, Dietrerich et al. structurent en deux grandes classes l'espace des constituants sujets à adaptation : les fonctions générales (comme la correction d'erreur et l'aide), et l'interaction qui regroupe l'adaptation de la présentation en entrée comme en sortie, et la simplification des tâches utilisateurs par le biais notamment de macrocommandes.

Modélisation et acquisition des connaissances. Avec cette dimension modélisation et acquisition des connaissances [Dieterich et al. 1993] donnent des indications sur l'espace solution.

Les modèles utiles au processus d'adaptation sont multiples, les plus courants comprenant le modèle de l'utilisateur — la majorité des systèmes font de l'adaptation à l'utilisateur —, les modèles des tâches, du dialogue, de l'application, enfin les métamodèles qui décrivent comment faire l'adaptation. Ces modèles nécessitent d'être alimentés en information. Il s'agit donc d'acquérir les connaissances qui leur sont utiles.

[Dieterich et al. 1993] distinguent la configuration de départ des connaissances, les acteurs et l'instant d'acquisition des connaissances. Dans la configuration de départ, les connaissances peuvent être inexistantes, ou bien s'appuyer sur des stéréotypes ou encore correspondre à des connaissances spécifiques à la cible (suite à transcription des réponses à des enquêtes préalables) [Crow et Smith 1993]. Les acteurs de l'acquisition peuvent être l'utilisateur ou le système, et l'acquisition peut avoir lieu avant, pendant et après une session d'utilisation.

Dans l'approche par stéréotype, des classes cibles (par exemple des groupes d'utilisateurs) sont déterminées par avance. Soit la cible effective se projette dans l'une de ces classes (il existe alors une interface adaptée), soit la cible ne se projette pas ou ne se projette que partiellement dans le stéréotype (et l'IHM produite n'est pas ou n'est que partiellement adaptée).

Dans une approche centrée individu, le système a un point de vue sur la cible et l'interface est adaptée selon ce point de vue. Il existe une interface adaptée à la cible si le système est capable de résoudre le système de contraintes qui représente la cible. Cette approche ouvre de nombreux problèmes sur la résolution des contraintes et la qualité de l'interface produite alors que dans une approche par stéréotype, il est possible de vérifier à la conception que les interfaces existent et sont de qualités. En effet dans une approche par stéréotype, toutes les contraintes sur la cible sont connues et résolues à la conception.

Niveaux d'adaptation. Pour chaque constituant adapté (fonctions générales et interaction), [Dieterich et al. 1993] distinguent cinq niveaux d'adaptation possibles, mais sans les expliciter formellement : les niveaux lexical, syntaxique, sémantique, tâche et but. Berthomé-Montoy propose seulement trois niveaux (syntaxique, heuristique et sémantique) mais en fournit une illustration technique concrète. Nous les présentons maintenant.

Les niveaux d'adaptation de Berthomé-Montoy

Dans ses travaux sur la génération d'interfaces adaptatives, Berthomé-Montoy vise comme objectif (au sens de AID) l'amélioration des perfomances de l'utilisateur et sa satisfaction [Berthomé-Montoy 1995]. Concernée par la mise en œuvre d'IHM adaptatives à l'utilisateur, elle distingue les niveaux syntaxique, heuristique et sémantique. Ces niveaux peuvent se voir comme un affinement de l'espace "déclencheurs-recommandations" de AID .

Niveau syntaxique. Ce niveau concerne les habitudes et goûts de l'utilisateur en matière de présentation et, pour les commandes, son expérience. La détection des préférences de présentation s'appuie sur des hypothèses comme "si l'utilisateur exécute les mêmes actions en début de session sur des éléments de présentation, c'est pour recréer un environnement qui lui convient". Ainsi, par un système d'historique, le système est-il capable de reconstruire l'apparence choisie par l'utilisateur. Pour le niveau d'expérience, chaque commande se voit associée d'un attribut (débutant, intermédiaire, expert) dont la valeur est déterminée à partir du taux de succès/erreur d'utilisation de cette commande. Selon le niveau d'expertise de l'utilisateur pour une commande donnée, le système procédera par étape (guidage du débutant) ou pour l'expert, présentera en un coup tous les paramètres nécessaires à la commande.

Le niveau heuristique. Ce niveau a trait au niveau tâche et à l'amélioration des performances de l'utilisateur. Ici, Berthomé-Montoy propose deux services : 1) le calcul dynamique des valeurs par défaut au niveau de chaque concept du domaine manipulé par l'utilisateur et 2) la détection de séquences d'actions fréquentes en vue de proposer à l'utilisateur la création de macrocommandes avant que celui-ci ait eu le temps de trouver fastidieuse la répétition de ces séquences.

Le niveau sémantique. Ce niveau concerne la détermination du but de l'utilisateur à partir de ses actions et de l'arbre de tâches en vue de proposer une aide adaptée : expliquer à l'utilisateur ses erreurs, lui expliquer le fonctionnement des commandes et l'aider à atteindre son but en découvrant les possibilités du logiciel.

Les taxonomies présentées jusqu'ici sont résolument orientées par l'objectif d'adaptation dynamique à l'utilisateur. Autrement dit, elles supposent fixes les attributs "plate-forme" et "environnement" des cibles[1] visées. Nous présentons maintenant des taxonomies centrées IHM multicibles.

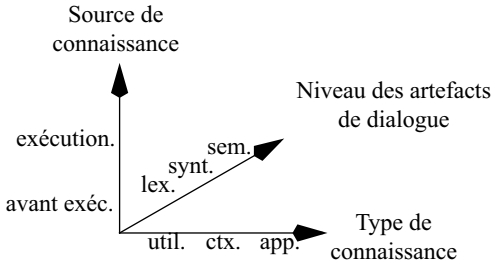
1.2. TAXONOMIES CENTRÉES IHM MULTICIBLE

Les taxonomies concernées par l'adaptation multicible sont récentes, motivées par la diversité des contextes d'interaction. Elles s'appuient, mais en les simplifiant, sur les taxonomies centrées utilisateurs. Nous avons retenu les travaux de Stephanidis et al. et de Kobsa et al. qui offrent des vues complémentaires sur l'espace problème des IHM multicibles.

Taxonomie de Stephanidis et al.

Comme le montre la figure 2, [Stephanidis et Savidis 2001] structurent l'espace problème de l'adaptation en trois axes : la source de connaissance, le niveau des artefacts de dialogue et le type de connaissances.

Figure 2. Espace de classification de [Stephanidis et Savidis 2001].



Source de connaissance. La source de connaissance (*source of knowledge*) correspond à la notion d'instant d'acquisition des connais-

1. Une cible, nous le rappelons, se définit par le triplet <classe d'utilisateurs, plate-forme, environnement>, Cf. Chapitre d'Introduction .

sances de [Dieterich et al. 1993]. La connaissance est soit connue avant l'exécution (information stéréotypée), soit déterminée en cours d'exécution (information individuelle). Avec l'instant d'acquisition, Stephanidis fait la distinction entre adaptabilité et adaptativité : Selon lui, un système adaptable est capable d'auto-adaptation aux connaissances initiales connues avant l'exécution, tandis qu'un système adaptatif est capable d'auto-adaptation aux connaissances acquises à l'exécution. Ces définitions sont à distinguer de l'usage général qui lui, s'appuie, non pas sur l'instant d'acquisition des connaissances, mais sur l'intervention explicite ou implicite de l'utilisateur final (cf. annexe "Définitions" à la page 207).

Niveau des artefacts de dialogue. Le niveau des artefacts de dialogue (*level of dialogue artefacts*) désigne le niveau d'interaction concerné par l'adaptation. Il s'agit d'un sous-ensemble des niveaux et des constituants adaptés de Dieterich et al., mais comme eux, Stephanidis se contente de fournir des définitions par l'exemple. Il s'agit des niveaux lexical, syntaxique et sémantique de l'interaction. Les attributs de présentation (reconfiguration spatiale, changement de couleurs, etc.) relèvent du niveau lexical. Passer d'un style "objet d'abord puis fonction" comme en manipulation directe, au style langagier, "fonction d'abord puis objet", concerne le niveau syntaxique de l'interaction. Le niveau sémantique est illustré par le changement de métaphore d'interaction pour la présentation et la manipulation des concepts et fonctions du domaine [Savidis et Stephanidis 1998]. Notons que, parmi les constituants adaptés (au sens de Dieterich), Stephanidis et al. ne considèrent que l'interaction, passant sous-silence l'adaptation des fonctions générales (aide et corrections d'erreur).

Type de connaissance. Le type de connaissance (*type of knowledge*) correspond aux déclencheurs de AID ou encore aux modèles de Dieterich et al. : ces connaissances déclenchent ou dirigent le processus d'adaptation. [Stephanidis et Savidis 2001] considèrent les caractéristiques de l'utilisateur mais aussi, il convient de le souligner, le contexte d'interaction. Le contexte d'interaction, on le rappelle est un couple composé d'une description de la plate-forme via laquelle se fait l'interaction et d'une description de l'environnement dans lequel cette interaction a lieu.

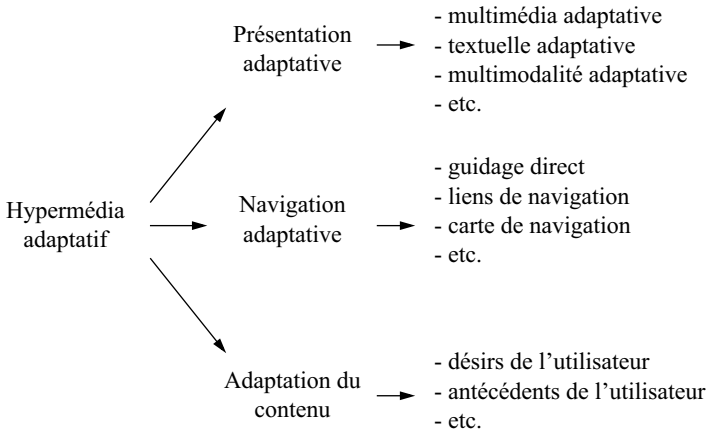
La taxonomie de Stephanidis et al. sert de point de départ à une méthodologie de conception d'IHM adaptatives, la conception d'IHM unifiée (*Unified User Interface Design*). A son tour, cette méthodologie a donné lieu au développement d'outils pour la construction d'IHM adaptatives dont le navigateur Web AVANTI. Les travaux de Kobsa et al. sur l'adaptation des hypermédias offrent une vue complémentaire sur le sujet.

Taxonomie de Kobsa et Brusilovsky

Au-delà des interfaces multimédia intelligentes [Maybury 1993] qui visent la génération automatique de représentations d'information [Roth et Hefley 1993], [André et al. 1993], [Feiner et McKeown 1993] pour un utilisateur donné et un contexte d'interaction donné, l'adaptation des systèmes hypermédias[1] fait depuis peu l'objet de recherches actives. Parmi les travaux récents, nous avons retenu la classification de [Brusilovsky 2001] et de [Kobsa et al. 2001]. Comme le montre la figure 3, cette classification repose sur trois facettes :

- l'adaptation de la présentation,
- l'adaptation de la navigation,
- l'adaptation du contenu.

Figure 3. Classification de l'adaptation dans les systèmes hypermédias. Tiré de [Brusilovsky 2001] et [Kobsa et al. 2001].



Ainsi, l'adaptation peut se faire soit en changeant la présentation (représentation d'une température par un thermomètre ou par une simple valeur numérique et son unité), soit en changeant la navigation (navigation par des liens ou au moyen d'une carte), soit en changeant le contenu en fonction des caractéristiques de l'utilisateur.

On constate que :

- Brusilovski et Kobsa se préoccupent peu de l'adaptation des tâches, l'objet document étant, sans conteste, la pièce maîtresse du problème[2].
- L'adaptation de la présentation est un cas d'adaptation du niveau lexical de la taxonomie de Stephanidis,
- L'adaptation de la navigation relève du niveau lexical (changement

1. Kobsa définit un système hypermédia comme un système interactif permettant à un utilisateur de naviguer dans un réseau d'objets hypermédias [Kobsa et al. 2001]. Un objet hypermédia est une entité composée d'éléments de différents types — texte, image, son, vidéo, application (Applet Java) ou éléments interactifs (menu, bouton, etc.). Le Web est le système hypermédia le plus connu où les pages Web constituent les objets hypermédias.

d'interacteur de navigation) et du niveau syntaxique (changement de la "manière" de naviguer),

- L'adaptation de la nature du contenu est un exemple d'adaptation du niveau sémantique.

1.3. SYNTHÈSE ET CRITIQUE

Les travaux et taxonomies analysés dans cette section offrent des éléments utiles à la structuration de l'espace problème de l'adaptation. Chacun d'entre eux privilégie un parti pris, par exemple, l'adaptativité à l'utilisateur chez Browne, Dieterich et Rautenberg, l'adaptativité des systèmes multicibles et multimédias chez Stephanidis, Kobsa et Brusilovski.

Si le projet AID, avec ses classes de métrique, soulève les bonnes questions en relation avec les étapes du processus de développement, les taxonomies présentées ici entremêlent plusieurs points de vue sans les expliciter. En particulier, les notions de "constituants adaptés" (fonctions générales et interaction de Dieterich) et les "niveaux d'adaptation" (lexical, syntaxique, etc.), définis par des exemples, sont non seulement sujets à interprétation mais couvrent à la fois l'effet de l'adaptation (qui concerne l'utilisateur final) et son impact sur les niveaux d'abstraction de l'interaction qui concerne, cette fois-ci, l'implémenteur du système et les outils de développement. Cette rapide critique justifie notre propre contribution en matière de taxonomie sur l'adaptation.

Nos objectifs, on le rappelle, concernent la création d'outils adaptés au développement d'IHM multicibles. En conséquence, notre proposition de classification comprend deux volets, l'un et l'autre centrés sur le logiciel : l'un permet de raisonner sur les IHM exécutables multicibles. L'autre structure l'espace des outils impliqués dans la production de telles IHM. Les deux sections qui suivent présentent successivement ces deux volets. Nous clôturons le chapitre au moyen d'exemples d'outils relevés dans la littérature et que nous analysons selon les dimensions de notre taxonomie.

2. Il n'existe souvent qu'une seule tâche : accéder à une information donnée — tâche de navigation. Ceci tend à évoluer avec la complexité des sites Internet. De plus en plus de pages Web peuvent-être considérées comme des applications interactives. Par exemple, les sites de commerce électronique pour lesquels nous retrouvons des patrons de tâches — rechercher un produit, manipuler son panier (consulter, ajouter, retirer), passer la commande et suivre la commande.

2. Une nouvelle taxonomie des IHM multicibles : un point de vue système

Nous avons adopté comme référence structurelle, la décomposition fonctionnelle du modèle d'architecture logicielle Arch [Bass 1991]. Nous l'avons déclinée ensuite selon les axes de facture plutôt système tirés des taxonomies précédentes.

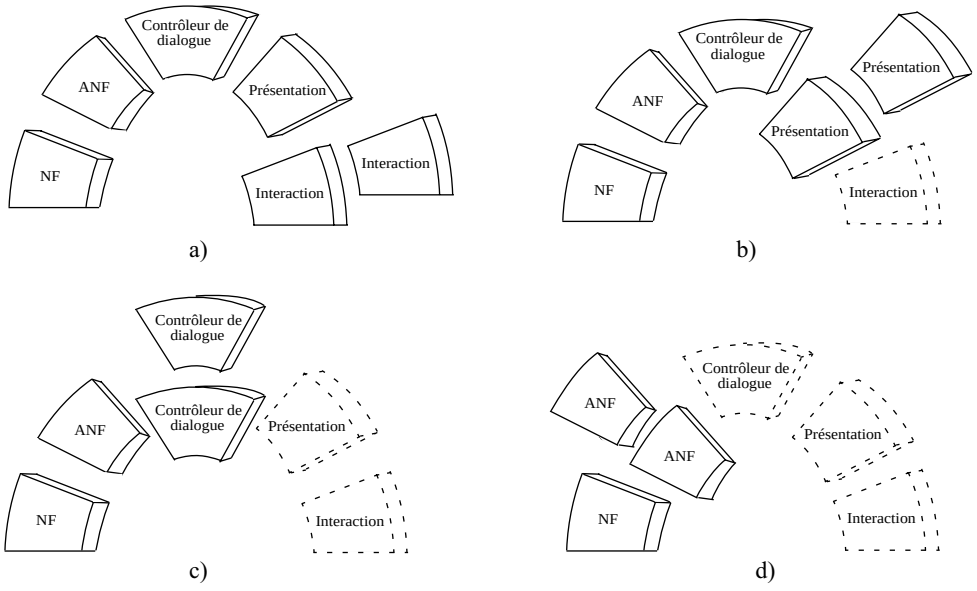
Nous présentons successivement les deux axes de notre taxonomie des IHM multicibles :

- Les constituants logiciels adaptés que nous calquons sur les niveaux fonctionnels de Arch,
- L'instant d'adaptation qui permet de distinguer les IHM exécutables précalculées des IHM exécutables dynamiques et hybrides.

2.1. CONSTITUANTS ADAPTÉS SELON ARCH

Comme le montre la figure 4, les constituants logiciels adaptés d'une IHM multicible correspondent aux niveaux d'abstraction fonctionnelle du modèle Arch. Ainsi, nous distinguons : l'adaptation des interacteurs physiques, celle des interacteurs de présentation logique, l'adaptation du contrôleur de Dialogue et celle de l'Adaptateur du Noyau Fonctionnel.

Figure 4. L'adaptation des IHM selon Arch. Les composants en pointillés expriment le fait que l'adaptation décrite, peut avoir des répercutions dans ces niveaux.

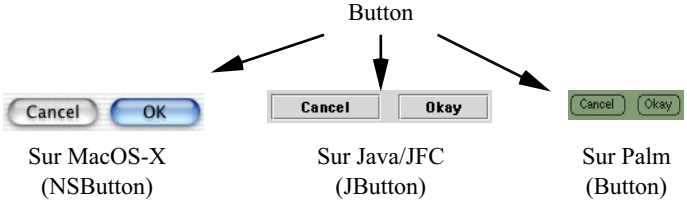


Adaptation des interacteurs physiques

L'adaptation des interacteurs physiques concerne l'adaptation du composant d'interaction physique de Arch. L'IHM s'adapte en utilisant les objets de la boîte à outils présents sur la, ou les, plates-formes cibles. Par exemple, sur Macintosh, l'IHM utilisera un NSButton et sur PC, un

WinButton. Ce type d'adaptation est utilisé dans Tk et Java/AWT avec le concept des *peers*[1]. **Avec ce type d'adaptation, la nature des interacteurs physiques est conservée, mais leur rendu peut être éventuellement distinct** (voir figure 4.a) La figure 5 montre l'exemple d'adaptation d'un bouton physique lorsque le système migre entre plates-formes (MacOS-X, Java/JFC et PalmOS).

Figure 5. Adaptation du niveau de l'interaction physique.



Adaptation des interacteurs logiques

L'adaptation des interacteurs logiques concerne le composant de présentation logique de Arch. Dans ce cas, l'IHM s'adapte en changeant de système représentationnel.

L'adaptation consiste à choisir parmi les interacteurs logiques disponibles ceux dont les capacités représentationnelles et fonctionnelles (ou navigationnelles[2]) sont équivalentes. **Avec ce type d'adaptation, les interacteurs sont de nature distinctes mais leurs capacités représentationnelles et fonctionnelles sont équivalentes** (voir figure 4.b). La figure 6 en montre des exemples.

La mise en œuvre de l'adaptation du niveau de la présentation logique peut s'appuyer sur la distinction classique entre Objets Interactifs Abstraits (OIA) et Objets Interactifs Concrets (OIC) [Vanderdonckt et Bodard 1993].

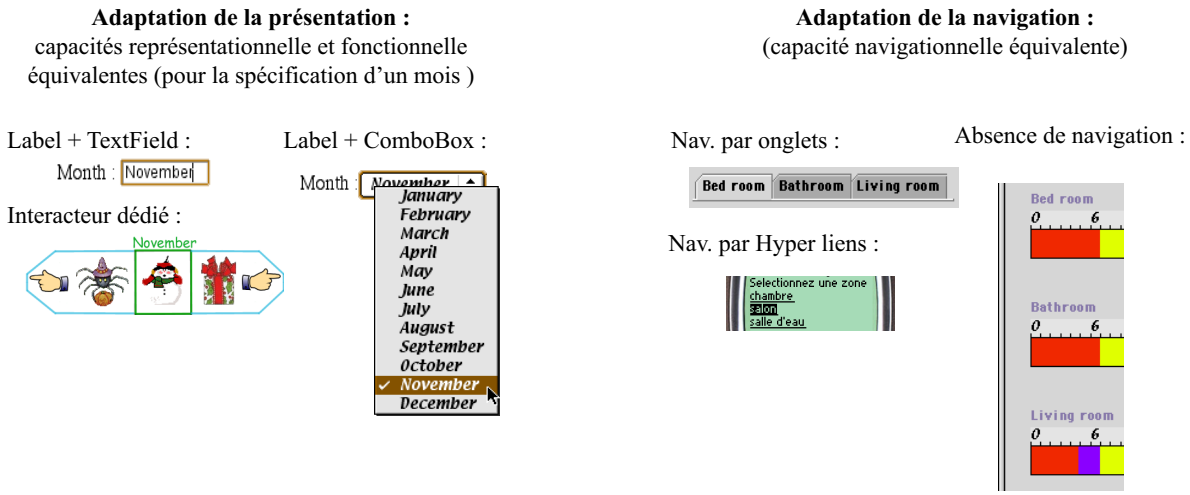
Ce type d'adaptation peut ajouter ou supprimer des tâches articulatoires (changement de navigation chez [Kobsa et al. 2001]). Nous considérons que ces tâches ne modifient pas le Contrôleur de Dialogue.

Adaptation du Contrôleur de Dialogue

Comme le montre l'exemple de la figure 7, une modification de ce type correspond à un changement de la structure du dialogue. **La nature des tâches reste inchangée, mais leur agencement diffère** (voir figure 4.c). Par exemple, passer d'un style de manipulation directe "sélectionner objet d'abord puis spécifier fonction" au style langagier,

1. Une boîte à outils graphique abstraite propose des interacteurs (le bouton par exemple). Il existe deux méthodes pour implémenter ce bouton : soit la boîte à outils utilise les primitives graphiques élémentaires (DrawRect, etc.) pour le dessiner, soit elle appèle le bouton correspondant, de la plate-forme cible. Ce deuxième principe correspond au principe des *peers* (pair).
2. La navigation est une tâche particulière que nous traitons différemment des autres tâches. Aussi des interacteurs spécifiques lui sont associés. Cf. le chapitre "Les descriptions dans ARTStudio", page 99.

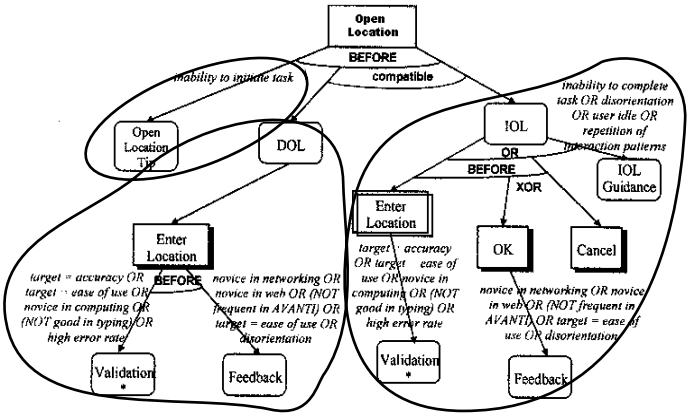
Figure 6. Adaptation du niveau de la présentation logique.



"spécifier fonction puis sélectionner objet", conserve les tâches mais change l'ordonnancement des tâches élémentaires. Les tâches polymorphiques de [Stephanidis et al. 2001] relèvent de ce type d'adaptation.

Figure 7. Adaptation du dialogue.

Exemple des tâches polymorphiques dans AVANTI [Stephanidis et al. 2001]. La même tâche est décomposée différemment selon l'utilisateur. Ici trois décompositions sont proposées.

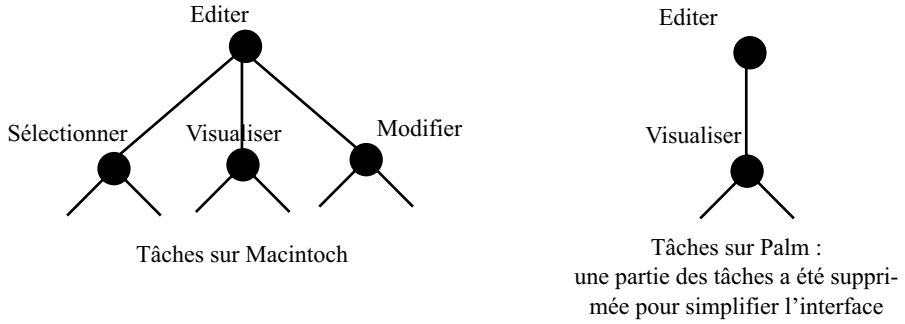


Adaptation de l'Adaptateur du Noyau Fonctionnel (ANF)[1]. Ce type d'adaptation correspond au changement de la nature des concepts et des fonctions exportés du Noyau Fonctionnel (l'application). **La nature des tâches et des concepts manipulés a changé** (voir figure 4.d). Cette forme d'adaptation est appliquée notamment lorsque les contraintes sont si fortes qu'il est nécessaire de supprimer des concepts ou des tâches. L'exemple de la figure 8 en fournit une illustration : une partie des tâches d'édition est supprimée lorsque l'application est accédée depuis un assistant personnel.

1. Le composant ANF, on le rappelle, sert de lieu d'adaptation naturelle entre le noyau fonctionnel et le reste de l'IHM. C'est le lieu privilégié pour pratiquer des réparations sémantiques par agrégation ou par filtrage de concepts et de fonctions du domaine. Ces modifications peuvent entraîner la modification des tâches.

L'adaptation de l'ANF est à rapprocher du niveau sémantique de [Stephanidis et Savidis 2001], et des niveaux (sémantique, tâches, buts) de [Dieterich et al. 1993]. [Zizi et Beaudouin-Lafon 1994] présentent une adaptation à ce niveau de composant Arch avec le zoom sémantique.

Figure 8. Adaptation de l'Adaptateur du Noyau Fonctionnel. Ses conséquences sur l'arbre de tâches et par suite sur le contrôleur de Dialogue.



Analyse Notre choix de calquer l'adaptation sur la décomposition fonctionnelle du modèle Arch offre plusieurs avantages :

- Un fondement solide. Le modèle Arch est une décomposition canonique qui fait référence dans la communauté de l'Ingénierie de l'Interaction Homme-Machine. Ses niveaux fonctionnels, aujourd'hui bien compris, constituent une base commune d'analyse.
- Un pont explicite entre adaptation et architecture logicielle.
- Un gain en précision et en clarté. L'effet de la propagation d'une adaptation peut être étudié selon un référentiel solide. Les niveaux lexical, syntaxique et sémantique des taxonomies citées précédemment ne permettent pas une telle analyse. En effet, une adaptation lexicale a des effets de bord sur les interacteurs physiques. Elle peut aussi en avoir au niveau de la présentation logique ou du contrôleur de dialogue. Par exemple, un changement de modalité, qualifié d'adaptation lexicale, a un impact sur les niveaux de Présentation logique et éventuellement d'Interaction physique.

2.2. INSTANTS D'ADAPTATION : INTERFACES PRÉCALCULÉES, DYNAMIQUES ET HYBRIDES

Si l'adaptation est souvent considérée à l'exécution, [Dieterich et al. 1993] identifient, nous l'avons vu, plusieurs intervenants possibles (concepteur, utilisateur, etc.). Dans une perspective système, ces acteurs correspondent à trois moments clés d'adaptation : au développement, à l'exécution et à l'installation :

Instants d'adaptation **Adaptation au développement.** L'adaptation est alors réalisée par les concepteurs et les codeurs qui ont prévu l'ensemble des cibles possibles. Par exemple les outils comme UIML [Phanariou 2000] ou AUIML (IBM) que nous décrivons dans la section 5, permettent la pro-

duction d'interfaces adaptées à différentes cibles, mais ces IHM ne peuvent pas s'adapter à l'exécution.

Adaptation à l'exécution. Comme son nom l'indique, l'IHM s'adapte ici à la volée. La boîte à outils MultimodalToolKit [Crease et al. 2000] discutée en section 4 permet un telle adaptation.

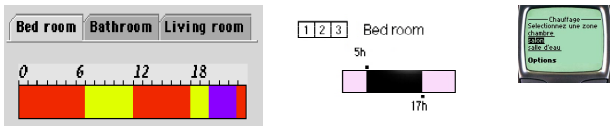
Adaptation à l'installation du logiciel . Le produit s'adapte et/ou est configuré pour correspondre à la cible donnée. Par exemple le finder de MacOS 9 de chez Apple Computer, peut-être configuré à l'installation pour proposer un sous-ensemble des fonctionnalités du système à travers des menus simplifiés (finder simplifié).

Ces trois instants d'adaptation nous conduisent à définir trois types d'IHM exécutables : les IHM précalculées, les IHM calculées dynamiquement et les IHM hybrides.

Types d'IHM multicible exécutable

IHM multicible précalculée. Une IHM multicible précalculée est le résultat d'une adaptation réalisée à la conception ou à l'installation. Les cibles possibles sont identifiées par avance et les IHM correspondantes sont créées et configurées en conséquence. La figure 9 montre un exemple d'IHM précalculée obtenue avec ARTStudio, notre outil de production d'IHM plastique que nous présentons en troisième partie de cette thèse. Nous parlons d'**adaptation précalculée**.

Figure 9. Adaptation précalculée. (IHM PC et Palm générées par ARTStudio) IHM permettant à un utilisateur de régler le chauffage des pièces de son domicile.

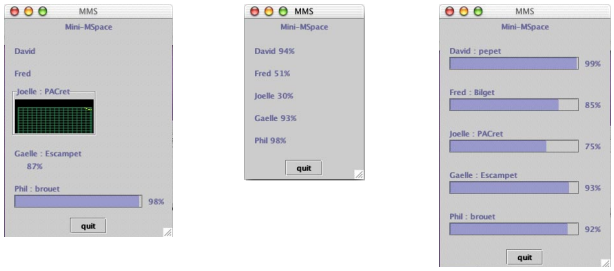


PC Palm WAP

IHM multicible calculée dynamiquement. Une IHM multicible calculée dynamiquement est élaborée en cours d'exécution dès qu'une condition d'adaptation se révèle vraie. Par exemple, dans la figure 10, l'IHM du mediaspace MMS*, qui montre le niveau d'activité des membres d'une communauté, est calculée à la volée en fonction de la taille de la fenêtre support. On parle d'**adaptation dynamique**.

IHM multicible hybride. Une IHM multicible hybride relève d'IHM précalculée et d'IHM calculée dynamiquement. On parle d'**adaptation hybride**. Par exemple, on dispose d'une IHM précalculée pour la classe des PDA et d'une IHM précalculée pour la classe des stations de travail. En sus, ces IHM s'adaptent pour correspondre parfaitement à la cible effective. En suivant notre exemple, l'IHM précalculée de la

Figure 10. Adaptation calculée dynamiquement. Application MMS* : l'IHM s'adapte en fonction de la dimension de la fenêtre.

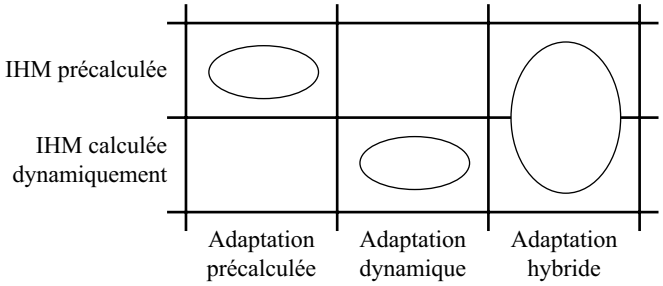


classe PDA s'adapte à l'exemplaire de PDA effectif (un PalmPilot, un PocketPC ou un Psion).

La figure 11 montre les relations entre les types d'IHM exécutables et les formes d'adaptations.

Notre nomenclature rappelle celle de [Dieterich et al. 1993] — avant, pendant et après une session d'utilisation — mais nous préférons les termes "précalculée" et "dynamique" pour caractériser la puissance des outils de conception. Nous dirons que «tel outil permet de concevoir des interfaces multicibles dont l'adaptation est précalculée» plutôt que «tel outil permet de concevoir des interfaces multicible dont l'adaptation est calculée avant exécution»!

Figure 11. Types d'IHM exécutables et type d'adaption.

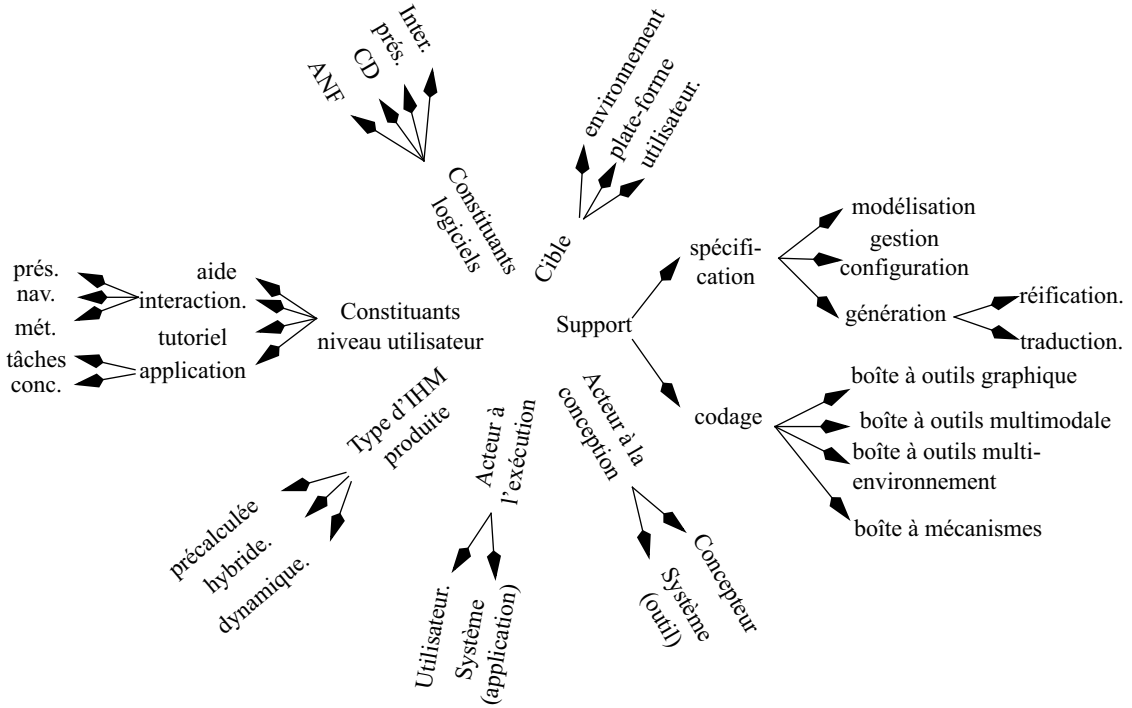


Selon le type d'IHM exécutable, il conviendra d'embarquer les mécanismes et modèles d'exécution idoines, mais aussi, nous l'avons vu avec les travaux de Dieterich et al., les moyens d'acquérir les connaissances sur les cibles. On se souvient des connaissances par stéréotype et centrées individu (cf. "Modélisation et acquisition des connaissances" à la page 17). Par définition, les IHM précalculées correspondent à une acquisition par stéréotype tandis que les IHM calculées dynamiquement supposent une acquisition centrée individu. Dans la suite de nos discussions, on se souviendra que "précalculé" implique "stéréotype" et que "dynamique" implique "individuel". Les IHM hybrides s'appuient sur un stéréotype au démarrage du système, puis virent à l'individuel à l'exécution.

3. Taxonomie des outils pour la production d'IHM multicibles

La "fleur" de la figure 12 représente de manière synthétique l'espace de classification des outils utiles à la production d'IHM multicibles. Elle comprend cinq axes : la nature de la cible couverte par l'adaptation (utilisateur, plate-forme, environnement), le type d'IHM exécutable produite par les outils (précalculée, dynamique, hybride), les constituants adaptés (du point de vue système et du point de vue utilisateur), les acteurs de l'adaptation (à la conception comme à l'exécution), intervention de l'outil dans le processus de conception comme support au développement (outil de spécification ou de codage). Concrètement, nous avons conservé les axes taxonomiques présentées plus haut lorsqu'ils correspondaient à un point de vue système. Nous détaillons maintenant chacun des éléments de la "fleur des outils pour IHM multicible".

Figure 12. La fleur des outils pour IHM multicibles.

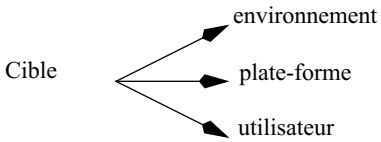


3.1. LA CIBLE Une cible, nous l'avons vu en introduction de ce mémoire, se définit par le triplet <utilisateur, plate-forme, environnement". Le domaine de valeurs de ce triplet définit la portée d'adaptation de l'outil, c'est-à-dire l'étendue des déclencheurs traités. Comme le montre la figure 13, nous ne détaillons pas plus avant les attributs qui caractérisent un environnement, une plate-forme et un utilisateur (bien que le nombre d'attributs et leur domaine de valeurs permettent d'affiner la puissance d'adaptation de l'outil). Avec notre outil ARTStudio, nous verrons des exemples

de tels attributs, mais nous pouvons déjà énumérer en exemple pour la plate-forme, les attributs suivants :

- charge machine (mémoire, puissance de calcul et 3D, etc.);
- charge réseau (bande passante, latence, etc.) ;
- dispositifs d'interaction physique disponibles, leurs capacités et coûts d'utilisation (coût utilisateur et computationnel) etc.

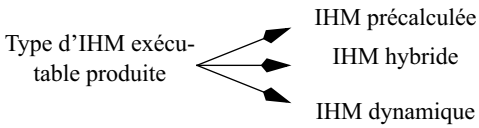
Figure 13. La cible traitée par l'outil.



3.2. TYPE D'IHM MULTICIBLE PRODUITE

Cet axe reprend la classification justifiée dans la section 2 : IHM précalculée, IHM dynamique et IHM hybride (voir figure 14).Le type d'IHM multicible produite mesure le degré de dynamicité que l'outil est capable de couvrir.

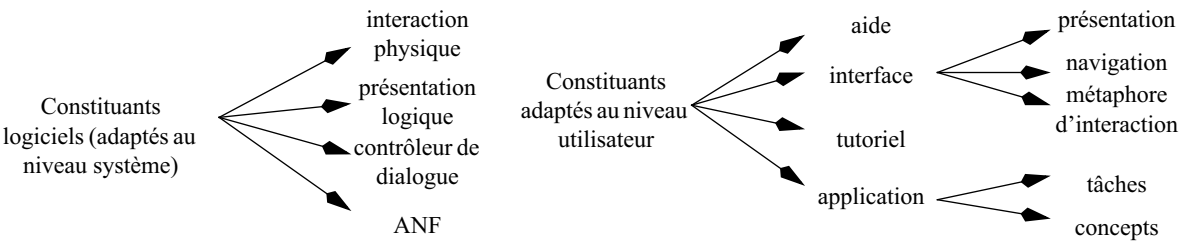
Figure 14. Type d'interface produite par l'outil.



3.3. CONSTITUANTS ADAPTÉS

Les constituants adaptés traduisent, comme dans la taxonomie de Dieterich et al., l'effet de l'adaptation tant du point de vue système que du point de vue utilisateur. Du point de vue système, les constituants adaptés désignent les composants logiciels marqués par l'adaptation. Ce sont les niveaux fonctionnels du modèle Arch que nous avons justifiés dans la section 2. Du point de vue utilisateur, les constituants adaptés correspondent aux recommandations (au sens de Browne et al.), c'est-à-dire aux effets perceptibles à l'utilisateur. Comme le montre la figure 15, nous avons affiné cet axe en reprenant l'apport des taxonomies présentées dans la première section : aide et corrections d'erreurs, tutoriel, application et interface.

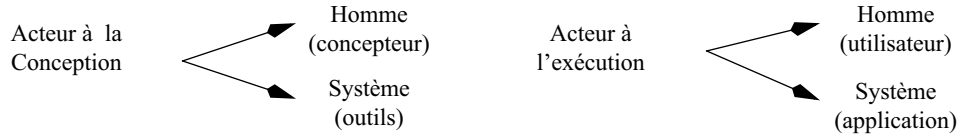
Figure 15. Constituants adaptés : points de vue utilisateur et système.



3.4. ACTEURS DE L'ADAPTATION

Les acteurs, nous l'avons vu avec la classification de Dieterich, sont les intervenants dans le processus d'adaptation. Ce processus comporte les 4 étapes de Dieterich (initiative, suggestion, décision, exécution) auxquelles nous ajoutons celle de l'évaluation. Pour un souci de simplicité, nous ne rentrons pas dans de tels détails dans notre classification.

Figure 16. L'acteur de l'adaptation.



3.5. SUPPORT AU DÉVELOPPEMENT

Le support au développement caractérise le niveau de confort des services fournis par l'outillage tout au long du processus de développement, depuis la conception jusqu'à la mise en œuvre. On distingue classiquement deux catégories d'outils[1] :

- Les outils de codage qui, en l'état des avancées techniques en matière d'IHM multicible, se résument aux bibliothèques de services réutilisables.
- Les outils de spécification qui couvrent les multiples besoins du Génie Logiciel. Considérant notre sujet sur les IHM multicibles, nous avons retenu trois classes d'outils de spécification : outils de modélisation, outils de génération et outils offrant la gestion de configurations.

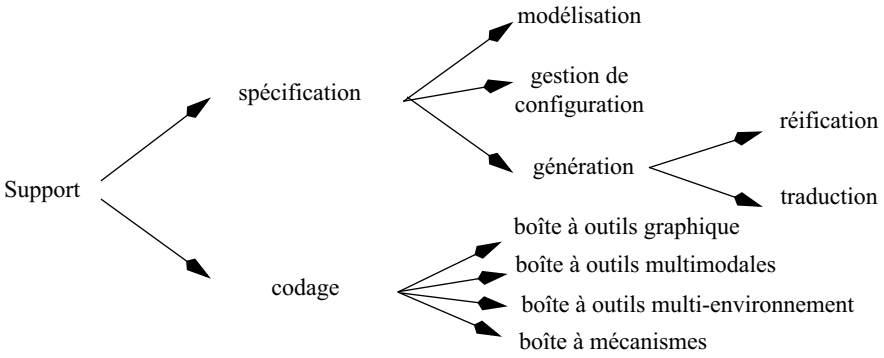
La figure 17 résume l'espace des outils considérés. Nous les passons brièvement en revue.

Outils de codage

Si l'on considère le cas général, les outils de codage sont nombreux. En relation avec les problèmes de l'adaptation, nous retenons quatre classes particulières de boîtes à outils : boîtes à outils graphiques abstraites, boîtes multimodales, boîtes pour les systèmes sensibles à l'environnement et boîtes à mécanismes.

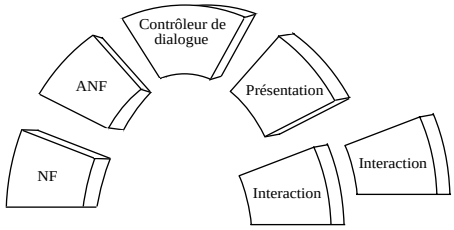
1. Les progrès aidant, la frontière entre les activités de conception et de mise en œuvre des IHM est de plus en plus floue. Par le passé, la distinction était claire. Par exemple, un arbre de tâches était spécifié de manière déclarative Cf, CTT [Paternò et al. 1997] : c'était de la conception. Puis, fondé sur cette description, on développait un contrôleur de dialogue en C++ : c'était de la mise en œuvre. Aujourd'hui, un outil d'aide à la conception peut générer une application. Dans le sens inverse, les outils de mise en œuvre deviennent de plus en plus abstraits (par exemple pour faciliter la portabilité, l'adaptation, leur utilisation !) et couvrent ou tendent à couvrir des aspects conception (par l'intégration de méta-informations, de niveaux d'abstraction dans le même langage). Ainsi, un logiciel de développement de pages Web WYSIWYG peut s'utiliser pour faire du maquettage en vue de comprendre le problème et pratiquer de l'évaluation formative [Hix et Hartson 1993] : c'est de la conception. Mais on peut aussi produire une IHM finale : c'est de la mise en œuvre. Un même outil peut donc couvrir plusieurs activités et étapes du processus de développement. Toutefois, un outil de codage et un outil de spécification offrent, du point de vue du développeur-concepteur, des niveaux de confort bien distincts.

Figure 17. Support au développement.



Boîte à outils graphique abstraite. Les boîtes à outils graphiques abstraites assurent la portabilité entre plates-formes. La cible n'est donc variable que dans sa dimension plate-forme et la couverture d'adaptation est celle du niveau interaction physique de Arch (adaptation des interacteurs physiques, cf. figure 18).

Figure 18. Boîte à outils graphique abstraite. Adaptation du niveau interaction physique dans Arch (cf. figure 4).

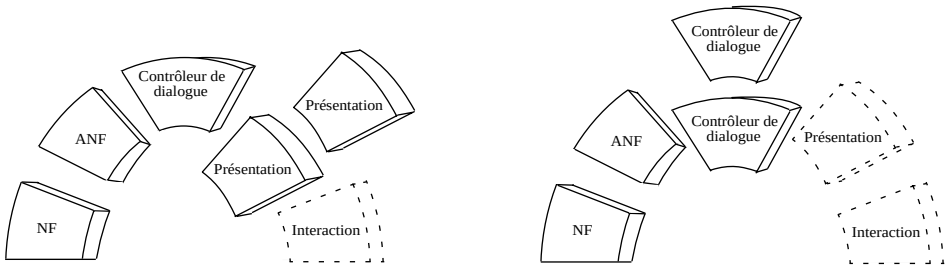


Boîte à outils multimodale. La multimodalité se définit comme l'utilisation de plusieurs modalités de manière alternée ou parallèle, de façon combinée ou redondante [Nigay 1994]. Du point de vue système, une modalité (qu'elle soit d'entrée ou de sortie) se définit par l'association d'un système représentationnel et d'un dispositif physique d'interaction [Nigay et Coutaz 1996].

Si à l'origine, la multimodalité était étudiée pour améliorer l'efficacité de l'interaction entre un utilisateur et un système, la multimodalité se voit aujourd'hui comme un facteur d'adaptation des IHM aux ressources physiques disponibles comme aux variations de l'environnement.

Comme le montre la figure 19, un changement de modalité implique un changement de présentation logique (si le système représentationnel est changé) et/ou du niveau d'interaction physique (si le dispositif d'interaction est changé).

Figure 19. Boîte à outils multimodale. Adaptation du niveau logique et éventuellement au niveau CD dans Arch (cf. figure 4).



Il existe que peu d'exemples de boîtes à outils multimodales conçues pour la mise en œuvre d'IHM multicibles. Les travaux de [Crease et al. 2000] constituent une exception que nous présentons en détail dans la section 4.

Boîte à outils multi-environnement. Comme pour les boîtes à outils graphiques qui offrent un modèle computationnel et des widgets d'utilité publique (menu, formulaire, bouton), l'objectif de ces boîtes à outils est d'offrir un modèle d'exécution et des composants réutilisables pour capter les phénomènes environnementaux et les restituer à l'application au niveau d'abstraction approprié en vue d'effectuer une adaptation. Dès lors, l'application peut décider de la nature des constituants à adapter (ANF, CD, présentation logique et/ou interaction physique).

La notion de contexteur de [Rey 2001] relève de cette tendance. Le Context-Toolkit [Dey 2000] que nous présentons en détail dans la section 4 est, à notre connaissance, le seul outil disponible de sa catégorie.

Boîte à mécanismes. Une boîte à mécanismes couvre les aspects logiciels qui interviennent, non pas dans la réalisation de l'IHM multicible proprement dite, mais qui implémentent une ou plusieurs des étapes du processus d'adaptation : l'initiative, les propositions, le choix, l'exécution, l'évaluation.

Les outils basés sur XSL/XSLT [Clark 1999], utilisés notamment pour transformer des pages HTML en pages WML, incluent des mécanismes de traduction d'arbres. Les moteurs d'inférence, les systèmes de résolution de contraintes comme dans Cicero [Arens et Hovy 1995], sont d'autres exemples de tels mécanismes. Dans cette catégorie d'outils, il faut compter les modèles computationnels et architecturaux (client-serveur, souscription par abonnement, introspection, etc.) qui dépassent le cadre de notre étude.

Les quatre catégories d'outils de codage que nous venons de présenter viennent compléter ou servent de logiciel de base au développement d'outils de spécification.

Outils de spécification

Nous avons retenu pour notre sujet d'étude trois catégories d'outil de spécification : les outils de modélisation, les outils de génération et les outils de gestion de configuration.

Modélisation. La modélisation est une activité qui consiste à produire des représentations (dites modèles), éventuellement mathématiques, d'un phénomène ou d'un système. L'intérêt immédiat de la modélisation est de maximiser la réutilisabilité et d'augmenter la productivité en permettant, par exemple, la génération automatique de code [Bruno 1995]. La modélisation va souvent de pair avec la formalisation.

Formaliser, c'est produire une description non ambiguë offrant de facto un double avantage :

- Le modèle est utilisable par plusieurs personnes selon une interprétation unique,
- Le modèle peut être soumis à une vérification automatique de propriétés (par exemple, démontrer que la génération fonctionnera toujours).

Inversement, la formalisation se heurte à deux difficultés majeures expliquant l'échec relatif des approches de type MB-IDE (*Model-Based Interface Development environmen*t) :

- La lourdeur de la spécification suscite le rejet de la plupart des développeurs et des concepteurs. C'est seulement dans des domaines très spécifiques comme les systèmes critiques pour lesquels le rapport coût/bénéfice est favorable, que les spécifications formelles sont utilisées avec succès.
- L'impossibilité de formaliser de manière réaliste, c'est-à-dire sans trop de simplifications, le comportement de l'utilisateur ou de l'environnement. En étant caricatural, il n'est possible de formaliser que des systèmes et phénomènes simples !

Aussi de nombreuses approches de modélisation ne sont pas formelles même si elles en donnent l'impression par l'utilisation de notations mathématiques.

Génération. Nous proposons d'appréhender le processus de génération selon deux axes. Tandis qu'une génération verticale produit, par réification, une interface pour une cible donnée, une génération horizontale fait une traduction, à un niveau d'abstraction donné, vers des

descriptions de même niveau d'abstraction, mais adaptées à une autre cible.

- **Réification**. L'ingénierie logicielle traditionnelle adopte un processus de production vertical — spécification des requis, conception générale, conception de l'architecture, [...], intégration&test, maintenance [Dix et al. 1998] — et procède principalement par affinements successifs en partant de modèles abstraits — par exemple, les spécifications externes — pour aboutir à des modèles concrets — le code exécutable. Ce processus de production a été largement suivi dans les générateurs du type *Models Based Interface Development Environment* (MB-IDE) [Szekely 1996]. C'est ce que nous appelons la génération/conception par réification : du plus abstrait au plus concret. Nous en verrons de nombreux exemples dans la section 5 avec notamment Mobi-D et Use-IT.
- **Traduction**. La génération par traduction consiste à utiliser en entrée un modèle représentant le système (l'IHM) à un niveau d'abstraction donné pour produire une représentation de ce même système (IHM) au même niveau d'abstraction. Par exemple la génération d'une interface graphique pour PalmPilot à partir d'une interface graphique pour PocketPC répond à un processus de traduction. Plusieurs implémentations du système XSL/XSLT [Clark 1999] proposent une adaptation par traduction en produisant une interface WML à partir d'une interface HTML.

Réification et traduction constituent les deux piliers directeurs du processus de production d'IHM multicible que nous présenterons au chapitre 3.

La gestion de configuration. La gestion de configuration ou Software Configuration Management (SCM) est le contrôle de l'évolution d'un système complexe [Estublier 2000]. Dans une conception visant la production multicible, la gestion de configuration et en particulier des différentes versions d'un modèle (le versionnement) en fonction de la cible devient importante.

La gestion de configuration se décompose en trois sous domaines [Estublier 2000] :

- La gestion d'un dépôt des composants logiciels (*Component Repository*) qui se traduit entre autres par une gestion de versions et de composition ;
- Le support à l'ingénieur pour le travail coopératif et le développement (par exemple aide à la compilation, dépendance entre fichiers,

débogage) ;
- Le support et contrôle du processus de conception.

Dans une optique de conception d'IHM multicibles, un SCM aidera le concepteur à :

- Partager les ressources afin de minimiser les efforts de conception et assurer la cohérence des informations communes ;
- Définir les relations entre les modèles du processus de conception (maîtriser les dépendances et éventuellement les minimiser) ;
- Mettre en avant les différences entre plusieurs versions d'un même modèle (par exemple, expliciter les différences entre le modèle des tâches correspondant à l'utilisateur expert et celui de l'utilisateur novice).

A notre connaissance, aucun outil d'aide au développement d'IHM multicibles ne couvre la gestion de configurations et le versionnement à tous les niveaux de la conception. Aussi, nous verrons dans le chapitre suivant les principes de description, factorisation et décoration. Ces principes représentent nos fondements théoriques pour la conception multicible. Nous en verrons ensuite une instanciation dans notre outil ARTStudio, dans la troisième partie.

Les deux sections qui suivent présentent respectivement les outils de codage et les outils de spécification en matière d'IHM multicible.

4. Outils de codage d'IHM multicible

Nous passons en revue les outils de codage selon les termes de notre taxonomie : boîtes à outils graphiques abstraites, boîtes à outils multimodales avec Multimodal Toolkit, boîtes à outils multi-environnement avec Context-Toolkit et mécanisme avec Seescoa XML.

4.1. BOÎTES À OUTILS GRAPHIQUES ABSTRAITES

Tk dans Tcl [Ousterhout 1994], GTK [GTK] et AWT/Swing dans Java [SUN/J2SE] sont des boîtes à outils graphiques abstraites. Elles diffèrent par le modèle d'exécution : Tcl/Tk est interprété, GTK est compilé et AWT/Swing/Java est précompilé en "byte code" puis interprété par une machine virtuelle.

Pour une classe de plate-forme donnée, Swing et Tk, avec la notion de *layouts manager*, permettent la reconfiguration dynamique du placement des interacteurs physiques selon des règles précalculées, voire décrites sous la forme de contraintes. Il s'agit d'une adaptabilité dynamique à la surface d'affichage.

Notons que la portabilité est une forme limitée d'adaptation à la plate-forme. En l'état, les boîtes à outils graphiques abstraites ne tolèrent que des variations réduites des attributs de plate-forme. En raison des différences de ressources (écran, mémoire, performances), une IHM programmée pour une machine de bureau ne peut être portée telle quelle sur PDA. Néanmoins, le "logiciel de base pour le multiplate-forme" se développe progressivement. Citons Java/AWT et toutes ses déclinaisons pour système embarqué (comme Waba, SuperWaba, Kada, KVM), VisualBasic (qui devrait être porté pour WinCE) et son homologue RealBasic (disponible sur MacOS 9, MacOS X et Windows), la bibliothèque Qt (disponible sous Unix, Windows, MacOS X et en cours de portage pour les PocketPC sous Linux).

La table 2 résume notre synthèse des boîtes à outils graphiques abstraites.

Table2. Synthèse des boîtes à outils graphique abstraite

Acteurs (phase de conception)	Sans objet
Support au développement	Outils de codage (boîte à outils graphique)
Constituants adaptés	Système : Interaction Utilisateur : Interface
Cible	Plate-forme
IHM exécutable	Dépendant de l'approche (par exemple AWT/Swing, Tk : dynamique)
Acteurs (phase exécution)	Système

4.2. MULTIMODAL TOOLKIT

Multimodal ToolKit [Crease et al. 2000] est une boîte à outils multimodale construite pour adapter l'interface de sortie d'un logiciel au contexte d'interaction. Elle permet une adaptation à la plate-forme et notamment aux dispositifs d'interaction disponibles (écran 1024*768 d'une station de travail ou 160*160 d'un PalmPilot), à la variation de leur charge d'utilisation (surcharge du dispositif sonore), ou à leur retrait ou ajout au système (retrait ajout d'une souris, d'un clavier). Elle permet également une adaptation au lieu d'utilisation de l'application.

Un apport important de ce travail est de proposer une architecture, pour la boîte à outils, construite de manière à séparer le comportement d'un interacteur, de son interface utilisateur d'entrée/sortie. De cette manière, l'interface peut s'adapter dynamiquement sans changement du comportement interne des interacteurs.

Une limite de ce système par rapport à nos objectifs est l'absence de gestion d'un modèle de présentation. Ce manque limite l'adaptation au seul changement d'interacteur, sans réorganisation de l'interface. En outre, l'utilisation d'un gestionnaire de présentation permet de distin-

guer le coût d'utilisation d'un interacteur donné, du coût de l'interface (qui correspond à l'utilisation d'un ensemble d'interacteurs, organisés selon un modèle de présentation).

Table3. Multimodal ToolKit en synthèse

Acteurs, phase conception	Le concepteur. Il construit l'interface à la main
Support au développement	Boîte à outils multimodale + boîte à mécanismes
Constituants adaptés	Point de vue système : OIA Point de vue utilisateur : Interface (la présentation)
Cible	Plate-forme (dispositifs d'interaction et capacité) et environnement
IHM exécutable	Dynamique
Acteurs, phase exécution	Système. Une dérive de cette adaptation est un changement explicite de la taille d'une fenêtre pour forcer l'adaptation. Dans ce cas, l'initiative de l'adaptation revient à l'utilisateur.

4.3. CONTEXT-TOOLKIT

Context-toolkit [Dey 2000], vient de l'analogie avec les bibliothèques graphiques de widgets. Comme pour les boîtes à outils graphiques qui offrent un modèle computationnel et des widgets d'utilité publique (menu, formulaire, bouton), Contex-Toolkit inclut un modèle d'exécution et des composants de base réutilisables.

Le constituant de base de l'architecture est un "context widget", composant autonome qui encapsule un capteur physique. Son rôle est de communiquer à un (ou plusieurs) serveur(s) ou interpréteur(s) les informations perçues en fonction des données reçues par le capteur. Les interpréteurs ont pour objectif de fournir une sémantique aux signaux qui leur sont transmis. Par exemple, un interpréteur de localisation de personne peut fournir comme informations :

- Personne A dans la salle B204
- Personne A dans le Bâtiment B de l'IMAG
- Personne A dans le Campus de Grenoble

Les serveurs collectent l'ensemble des signaux émis par les autres composants et font le lien entre les applications et les "contexts-widgets". Les serveurs ont également la charge de synthétiser les informations en informations de niveau d'abstraction supérieur. Cependant le mécanisme de synthèse n'est pas décrit et revient au programmeur.

Sur le plan technique, le système de communication entre les context-widgets et l'application se fait par souscriptions et la réaction aux souscriptions s'effectue sous forme de call-back (réaction).

Table4. Context-Toolkit en synthèse.

Acteurs (phase conception)	Sans objets
Support au développement	Boîte multi-environnement + mécanismes.
Constituants adaptés	Point de vue système : sans objet. Le context-toolkit ne fournit pas de mécanismes ou de composants décidant de la nature de l'adaptation. Point de vue utilisateur : sans objet
Cible	Environnement
IHM exécutable	Sans objets
Acteurs (phase exécution)	Sans objets

4.4. SEESCOA XML

L'objectif de Seescoa XML est de servir de langage de description non seulement d'IHM graphiques mais aussi de leur état d'exécution. Ce faisant, il devra être possible de faire migrer ces IHM dynamiquement entre plates-formes [Luyten et Coninx 2001]. Seescoa XML s'appuie pour cela sur le mécanisme de réflexion de Java qui permet d'analyser dynamiquement l'interface graphique et de générer la description XML correspondante. Cette description peut être ensuite envoyée sur une autre plate-forme pour y être instanciée en une interface graphique équivalente à l'originale et dans le même état. Le système est donc capable de cloner des IHM.

En l'état, [Luyten et Coninx 2001] ne gèrent qu'une adaptation à la boîte à outils graphique par le principe des OIA et OIC.

Table5. Seescoa XML en synthèse.

Acteurs (phase conception)	Sans objets
Support au développement	Boîte à mécanismes.
Constituants adaptés	Point de vue système : OIA, OIC Point de vue utilisateur : Interface
Cible	Plate-forme
IHM exécutable	Dynamique
Acteurs (phase exécution)	Système

5. Outils de spécification d'IHM multicible

5.1. UIML UIML est un langage déclaratif cherchant à masquer la diversité des plates-formes et des langages de développement — Swing, Waba, HTML, WML, VoiceXML, etc.— Il s'agit d'un langage simple pour lequel il existe aujourd'hui des compilateurs pour Java/JFC, PalmPilot, HTML, VoiceXML, et WML.

Mais avec UIML la spécification de la présentation est dépendante de la plate-forme. Si le développeur vise une IHM pour WML et pour Java/JFC, il se doit de produire deux spécifications distinctes : l'une utilisera les *tags* WML, la seconde les classes AWT/Swing avec des gestionnaires de présentation (*Layout manager*) Java.

Pour palier ce problème [Abrams et Phanariou 1999] font une double propositions :

- Un système d'abstraction des interacteurs selon le principe des OIA. Celui-ci fait partie des spécifications UIML 2.0 avec la combinaison des *tags* <peers> et <presentation>.
- Un système de gestion de la présentation qui fait encore l'objet de travaux (thèse de Mir Farooq Ali au Virginia Tech : http://vto-pus.cs.vt.edu/~abrams/uiml/).

Table6. UIML en synthèse.

Acteurs (phase conception)	Le concepteur. Il spécifie l'interface à la main. Seule l'exécution de l'adaptation est automatique puisque faite par chaque compilateur UIML vers {Java, WML, etc.}.
Support de développement	Outils de spécification utilisant un langage facilitant la mise en oeuvre pour des plates-formes disparates. Reste limité pour faire du multiplate-forme (en général nécessaire de faire une spécification UIML par plate-forme cible)
Constituants adaptés	Point de vue système : OIA Point de vue utilisateur : Interface (la présentation)
Cible	Plate-forme (interacteurs disponibles)
IHM exécutable	IHM précalculées (à raison d'une interface par cible)
Acteurs (phase exécution)	Sans objet puisque les interfaces sont précalculées

5.2. AUIML AUIML (*Abstract User Interface Mark-up Language*)[Merrick 2001] est un langage visant la génération automatique de code multiplate-forme (Dynamic HTML, Java Swing, Palm-Pilot). Il résout l'un des principaux problèmes de UIML puisque la spécification est indépendante de la plate-forme cible. En effet à aucun moment le programmeur décrit utiliser un JButton ou un "<href ..." dans sa description AUIML.

AUIML [Clark 2000] permet de décrire l'interface en termes :

- des éléments manipulés (données initiales, structures et données

complexes telles que les images ou du son) ;

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

Un autre point intéressant de 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. Par contre cet outil, propriété d'IBM, est très peu publié et n'est pas encore finalisé.

Table7. Synthèse de AUIML.

Acteurs (phase conception)	Concepteur : il spécifie l'interface à la main. Système : l'exécution de l'adaptation est automatique puisque faite par chaque compilateur AUIML
Support de développement	Langage de spécification pour la mise en oeuvre multiplate-forme
Constituants adaptés	Point de vue système : OIA et possiblement CD avec le gestionnaire de navigation Point de vue utilisateur : Interface (lprésentation et navigation)
Cible	Plate-forme (interacteurs disponibles, surface d'affichage)
IHM exécutable	IHM précalculées (à raison d'une interface par cible)
Acteurs (phase exécution)	Sans objet puisque les IHM sont précalculées

5.3. QTk

QTk [Grolaux et al. 2001] est un outil de spécification d'interface homme-machine construit au-dessus du langage Oz [Mozart]. Oz et QTk proposent au développeur un environnement de programmation rapide s'appuyant sur des principes issus de l'approche MB-IDE. En particulier QTk permet de bien séparer la présentation d'une interface, des données à présenter. Ainsi il est aisé d'associer plusieurs interfaces graphiques à la même application.

Ce principe est utilisé dans l'application FlexClock [Grolaux 2000] pour construire très simplement une interface graphique faisant de l'adaptation dynamique en fonction de la surface d'affichage. QTk propose le langage d'expression des différentes interfaces et les mécanismes de choix. Par contre, le concepteur doit décrire manuellement les différentes vues possibles.

Table8. Synthèse de la boîte à outils graphique QTk

Acteurs (phase de conception)	Construction manuelle des différentes interfaces
Support au développement	Outil de modélisation, et boîte à outils graphique et de mécanismes.
Constituants adaptés	Système : Interaction et Présentation Utilisateur : Interface

Table8. Synthèse de la boîte à outils graphique QTk

Cible	Plate-forme
IHM exécutable	Hybride
Acteurs (phase exécution)	Système (choix des interfaces)

5.4. HOMER UIMS

HOMER [Savidis et Stephanidis 1998] est un générateur d'IHM à deux utilisateurs (*dual user interface*), l'un d'entre eux étant atteint de cécité. Une IHM construite avec HOMER doit satisfaire les requis suivants : l'accès aux deux utilisateurs doit être simultané, les besoins des deux parties doivent être satisfaits (quitte à utiliser des métaphores d'interaction ou une structure d'IHM différentes), à tout moment la sémantique (concepts et fonctions) doit être identique pour les deux utilisateurs.

HOMER inclut un langage de spécification et utilise les notions de méta-intéracteur (*meta-widget*)[Wise et Glinert 1995] et de métaphore d'interaction pour produire une IHM. Le langage est interfacé avec une boîte à outils graphique, pour l'interaction visuelle, et une boîte à outils multimodale, pour l'interaction non visuelle. Il permet la spécification des présentations et du dialogue.

Table9. HOMER en synthèse.

Acteurs (phase de conception)	Le concepteur. Il spécifie l'interface à la main
Support au développement	Langage de spécification facilitant la mise en oeuvre multi-utilisateurs. Ce langage permet au concepteur de spécifier des métaphores d'interaction. Il est connecté à une boîte à outils graphique et multimodale.
Constituants adaptés	Point de vue système : OIA + CD (changement de métaphore => changement CD) Point de vue utilisateur : Interface (présentation et métaphore d'interaction)
Cible	Utilisateur (Cécité ou non)
IHM exécutable	Précalculées. Plusieurs IHM distinctes accessibles simultanément
Acteurs exécution	Sans objet puisque les IHM sont précalculées et accessibles en même temps.

5.5. MOBI-D

MOBI-D [Eisenstein et al. 2001], qui s'inscrit dans la lignée de Mecano [Puerta 1996], est un outil de type MB-IDE et vise la conception et la génération d'IHM multicibles. Il permet une adaptation de la présentation.

Dans cette optique, MOBI-D se fonde sur une modélisation des plates-formes cibles, du modèle de tâches et de différents modèles de présentation. Le système génère les interfaces graphiques en sélectionnant les interacteurs et une structure de présentation parmi un ensemble de structures de présentation prédéfinies. Il génère automatiquement le contrôleur de dialogue.

- Le choix des interacteurs est réalisé par le composant TIMM (*The Interface Model Mappe*r) [Eisenstein et Puerta 2000]. TIMM utilise

un arbre de décision pour sélectionner un interacteur logique (AIO) en fonction du concept du domaine à représenter (ex : String => champ texte). Ensuite l'interacteur logique et concrétisé en interacteur physique (CIO) en fonction de la plate-forme (ex en Swing : champ texte => JTextField).

- Le système utilise le concept de fenêtre logique et d'unité de présentation [Vanderdonckt 1997] pour la définition des modèles de présentation. La présentation est construite en fonction de l'arbre de tâche, d'un modèle de présentation choisi et des contraintes de surface d'affichage.

Table10. MOBI-D en synthèse.

Acteurs (phase conception)	système et concepteur : génération assistée par le système
Support de développement	Outils de spécification : modélisation (plate-forme, présentation, tâches), génération par réification
Constituants adaptés	Point de vue système : Navigation et présentation (OIA) Point de vue utilisateur : Interface (présentation)
Cible	Plate-forme (interacteurs physiques et surface d'affichage)
IHM exécutable	IHM précalculées.
Acteurs (phase exécution)	Sans objet puisque les IHM sont précalculées

5.6. USE-IT

USE-IT [Akoumianakis et Stephanidis 1997] est un outil de conception et génération d'interfaces supportant une conception multi-utilisateur. Il vise une adaptation à des classes d'utilisateurs prédéfinies, pouvant être des handicapés. Le système, visant initialement le multi-utilisateur, tient compte de la plate-forme cible. Nous le classons donc comme permettant une conception multiplate-forme.

Cet outil est conçu dans le but de représenter, manipuler et interpréter des connaissances pour la génération multicible. Il se fonde sur la spécification d'un modèle de l'utilisateur, des tâches et de la plate-forme (dispositifs d'entrée/sortie et interacteurs). Si la conception et la génération touchent les trois niveaux d'interaction (sémantique, syntaxique et lexical), les règles d'adaptation visent le niveau lexical. Aussi il s'appuie sur le principe des OIAs (objets interactifs abstraits) pour l'élaboration des règles d'adaptation.

Table11. Synthèse de USE-IT.

Acteurs (phase de conception)	Le concepteur (spécification) et le système USE-IT (génération automatique).
Support de développement	Outil de spécification. Modélisation (utilisateur, plate-forme et tâches). Génération par réification
Constituants adaptés	Point de vue système : OIA (adaptation du CD possible lors d'une adaptation de la métaphore) Point de vue utilisateur : Interface : la présentation
Cible	Utilisateur mais gestion du multiplate-forme (interacteurs et dispositifs d'entrée/sortie)

Table11. Synthèse de USE-IT.

IHM exécutable	IHM précalculées.
Acteurs (phase exécution)	Sans objet puisque les interfaces sont précalculées .

5.7. CICERO

Arens et Hovy s'intéressent à la difficulté croissante de développer des systèmes multimédia en raison de la diversité croissante des plates-formes. La taille du code et les coûts liés à la programmation d'une interface graphique les dirigent vers une approche d'adaptation dynamique de l'IHM. Dans cette optique, Arens et Hovy proposent le système Cicero [Arens et Hovy 1995], (système non implémenté dans son ensemble), permettant de faire de l'adaptation dynamique.

L'adaptation dynamique est organisée autour d'un médiateur (l'*Interaction Manager*) qui utilise cinq modèles de type déclaratif : un modèle de la plate-forme qui décrit les médias présents (caractéristiques, capacités), un modèle des concepts du domaine, un modèle des tâches (tâches proposées par le système et les buts de l'utilisateur), un modèle du dialogue (structure) et un modèle de l'utilisateur final (capacités, préférences). Ces modèles sont spécifiés par le concepteur au moyen des éditeurs adéquats.

Table12. Cicero en synthèse.

Acteurs (phase de conception)	Concepteur : il ne décrit pas l'interface, par contre il modélise un ensemble d'information qui sera utilisé par le système pour faire l'adaptation à l'exécution.
Support au développement	Spécification : modélisation (plate-forme, tâches, dialogue, concepts du domaine et utilisateur) Codage : boîte à mécanismes (médiateur intégré dans l'application générée).
Constituants adaptés	Point de vue système : niveau présentation Point de vue utilisateur : Interface (la présentation)
Cible	Plate-forme (médias disponibles) et Adaptateur de Noyau Fonctionnel (données à représenter, tâche à réaliser)
IHM exécutable	Dynamique : le système propose une architecture d'adaptation dynamique
Acteurs (phase exécution)	Système

5.8. AUTRES UIMS

Dans leur état de l'art, [Dieterich et al. 1993] classent de nombreux systèmes. Reprenons les caractéristiques des UIMS UIDE et PODIUM.

UIDE (*User Interface Design Environment*) [Sukaviriya et Foley 1993] est un UIMS permettant la génération automatique d'applications avec une aide adaptative. Il utilise des connaissances sur l'application, les tâches et les utilisateurs pour concevoir automatiquement une application capable de contrôler son exécution et fournir une aide adaptée. PODIUM (*Personalized Designer: an Intelligent User-Interface Manager*) [Sherman 1993] est un UIMS intégrant des connaissances sur les utilisateurs pour la génération automatique d'interfaces. Les interfaces

générées sont des applications de physique devant s'adapter à la connaissance de l'utilisateur physicien.

Table13. Synthèse de UIDE et PODIUM. Adapté de [Dieterich et al. 1993]

	UIDE	**PODIUM**
Acteurs (phase de conception)	Génération automatique par réification	Génération automatique par réification
Support de développement	Spécification : modélisation (utilisateur, tâches, application)	Spécification : modélisation (utilisateur, dialogue)
Constituants modifiés	Aide	Présentation
Cible	Utilisateur	Utilisateur (expérience)
IHM exécutable	Dynamique	Dynamique
Acteurs (phase exécution)	Système et utilisateur (initiative)	Système ou utilisateur ou système et utilisateur (décision)

5.9. XIML

XIML (eXtensible Interface Markup Language) est un langage de description d'interface graphique pour différentes plates-formes. L'objectif de ce langage est d'aider au développement d'interfaces multiplates-formes en diminuant le temps de développement et en minimisant les incohérences entre plates-formes. Dans ce but, il couvre tout le cycle de conception d'une interface en permettant la description :

- des aspects abstraits d'une interface (tâches, concept du domaine, utilisateur) ;
- des aspects concrets d'une interface (présentation, dialogue) ;
- des relations entre les éléments décrits (une présentation Y "présente" la donnée X).

Pour de meilleures diffusion et couverture, ce langage se veut ouvert et extensible :

- Il est du type XML et n'impose aucun outil de spécification. Il devra être possible d'utiliser les outils habituels et de faire un lien entre les données décrites par l'outil et les données décrites en XIML ;
- Il s'appuie sur une "métasyntaxe" pour faciliter son extension (il peut être considéré comme un métalangage). Ainsi il sera aisé de le connecter à des outils ou de le mettre à jour pour générer de nouvelles IHM.

Une description XIML est semblable à un dépôt centralisé de données. Elle est composée de trois piliers : le composant, l'attribut, la relation. Un composant est un élément décrivant les tâches, les concepts du domaine, les utilisateurs, les présentations ou le dialogue. Un attribut appartient à un composant. A partir d'un ensemble d'attributs, il est possible de décrire les caractéristiques ou les propriétés d'un composant. Une relation décrit un lien entre deux ou plusieurs composants.

Ce type de structure est à rapprocher avec les travaux de [Ducournau 1996] sur la formalisation de données Objets.

La couverture de la conception multiplate-forme est assurée par la description d'une présentation par plate-forme au sein de la description XIML. Eventuellement, il y aura à terme une description intermédiaire (indépendante de la plate-forme) qui sera mise en relation avec des présentations dépendantes. A charge aux outils d'assurer la description de la présentation intermédiaire et la génération des présentations dépendantes. Ce sera le but d'un outil comme MOBI-D.

Enfin XIML devrait offrir les ressources pour la gestion dynamique des interfaces en facilitant la réorganisation de la présentation, la gestion des préférences utilisateur pour la présentation, la distribution de présentation (un serveur envoie aux clients de nouvelles présentations).

La définition de ce langage est en cours de finalisation. Aucun outil n'est proposé pour gérer les différents niveaux de conception et la génération des IHM.

Table14. XIML en synthèse.

Acteurs (phase conception)	Le concepteur (dépendra des outils)
Support de développement	Langage de spécification : modélisation (devrait permettre : utilisateur, tâches, concept du domaine, présentation, dialogue). Il s'agit d'un méta langage donc ouvert à de nombreux modèles.
Constituants adaptés	Point de vue système : dépendra des outils de génération Point de vue utilisateur : dépendra des outils de génération
Cible	Initialement multiplate-forme mais a priori ouvert (attention : pas de modélisation de l'environnement prévue pour le moment)
IHM exécutable	Ouvert : dépendra des outils
Acteurs (phase exécution)	Objectif visé (mais dépendra des outils) : utilisateur (par préférence) et système

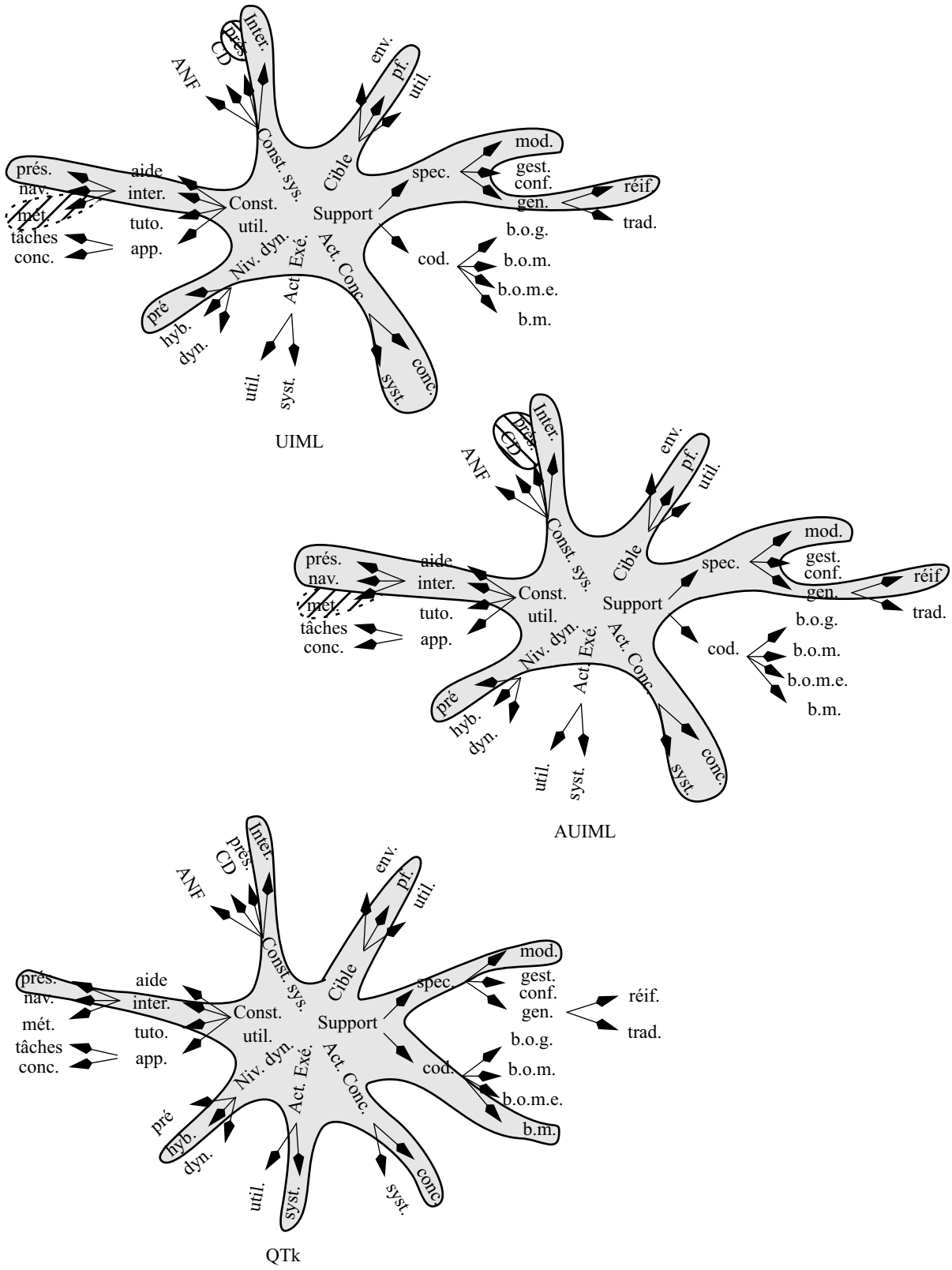
5.10. ET LES INDUSTRIELS?

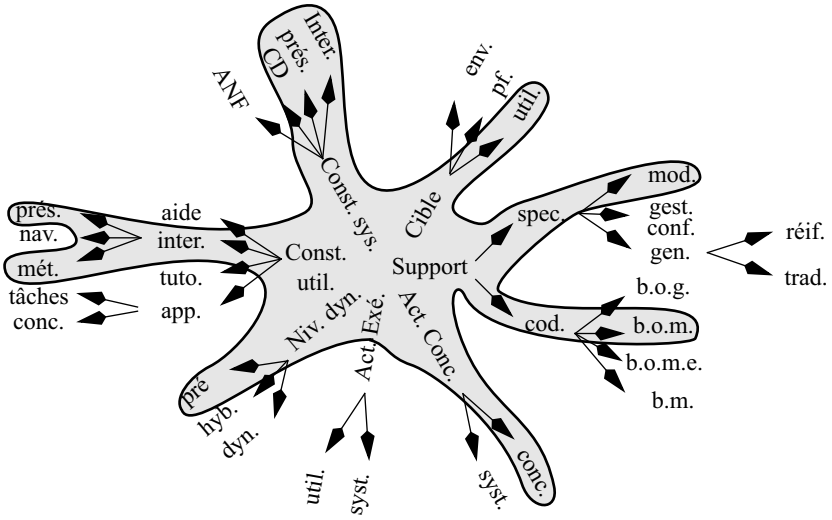
Le marché est très "juteux". Pour répondre à la mobilité et à la diversité des plates-formes, et assurer des services de commerce électronique, de B2B, etc., les industriels proposent de nombreuses solutions. Par exemple UIML est largement supporté par la société Harmonia fondée par le créateur de ce langage. IBM, Oracle, Wokup proposent des solutions ad hoc ou basées sur XSL/XSLT pour la traduction de page HTML vers WAP ou autres normes de description de page Internet pour téléphones portables et PDA (IBM WebSphere, Portal-To-GO, etc.). La société Palmware va même jusqu'à proposer le système MPS pour *Multi-Platform System*. Il s'agit d'une machine virtuelle, comme celle de Java, tournant sur PalmOS, PocketPC et Symbian Epoc (Psion). Nous ne décrirons pas plus ces systèmes dont le fonctionnement est très souvent mal documenté et qui reposent sur des principes déjà énoncés.

Il convient néanmoins de remarquer que les industriels proposent pour la plupart des approches par traduction plutôt que par réification. Il y a plusieurs explications à cela : outils plus simples à développer, rétroconception de l'existant et notamment des millions de pages html à récupérer et à transformer vers d'autres plates-formes.

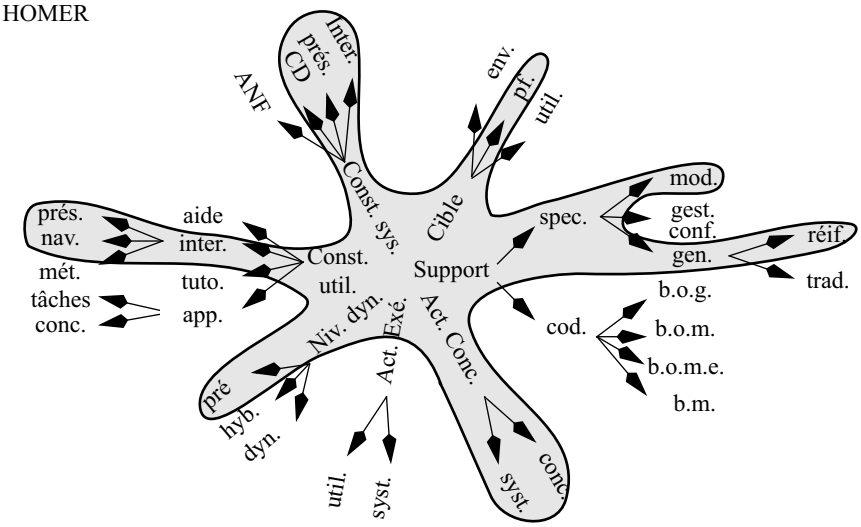
5.11. SYNTHÈSE

Figure 20. Projection des outils de conception sur l'espace de la fleur.

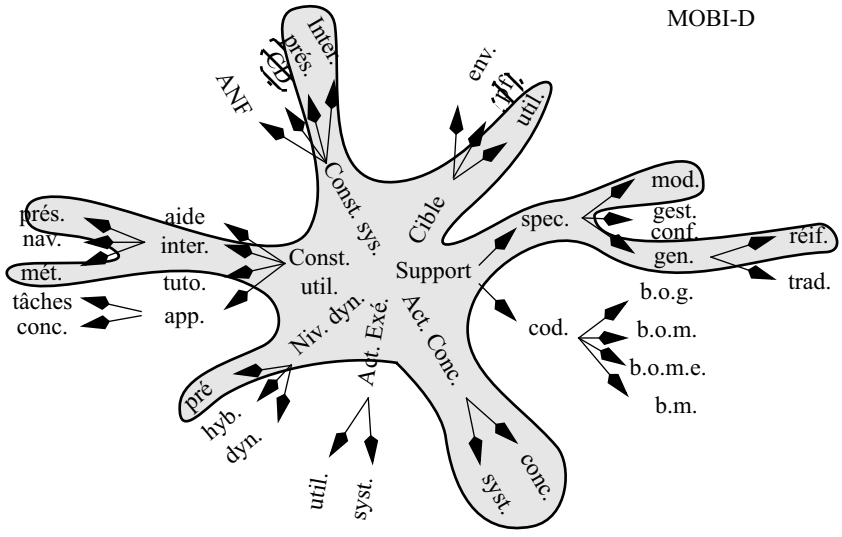




HOMER

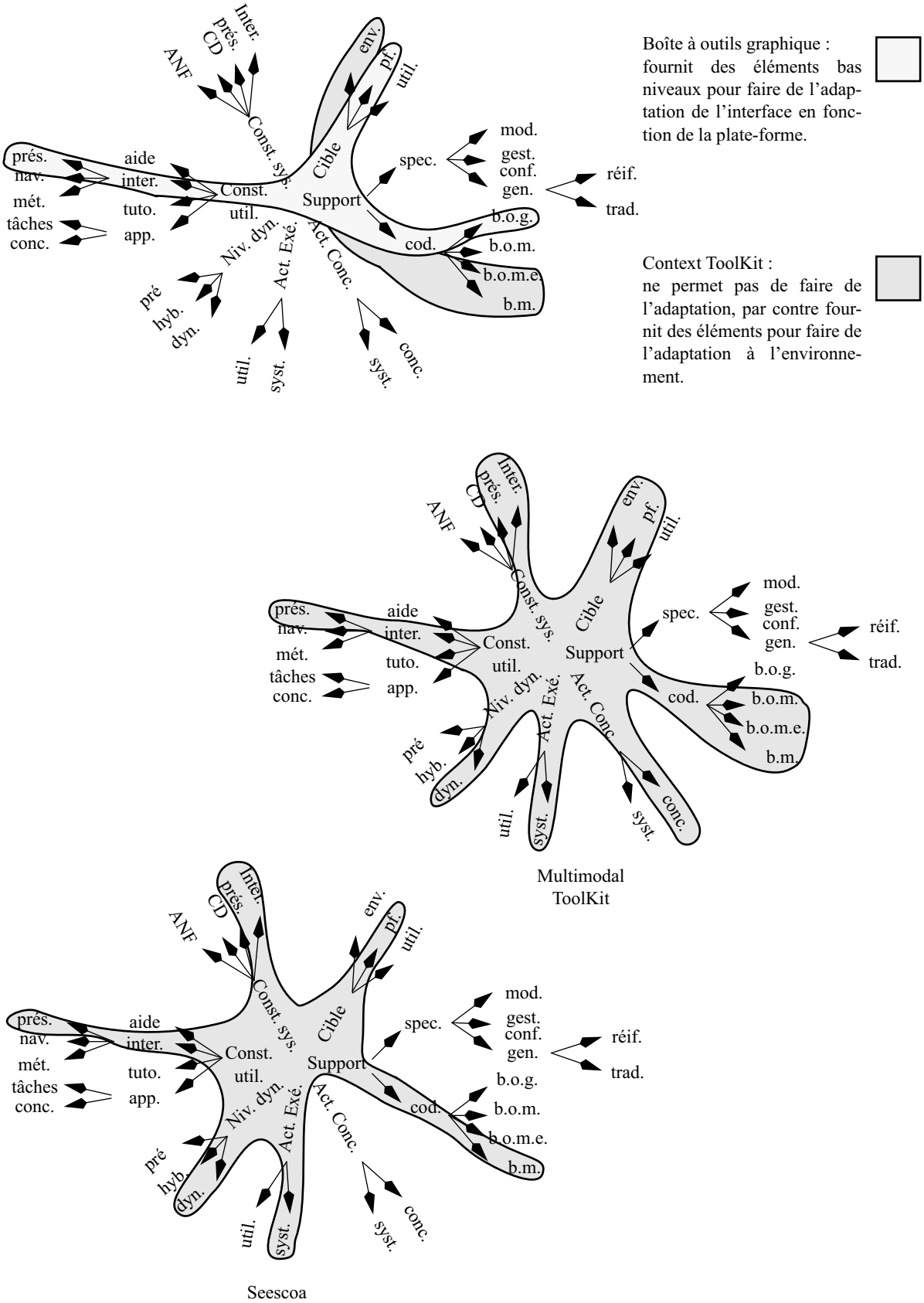


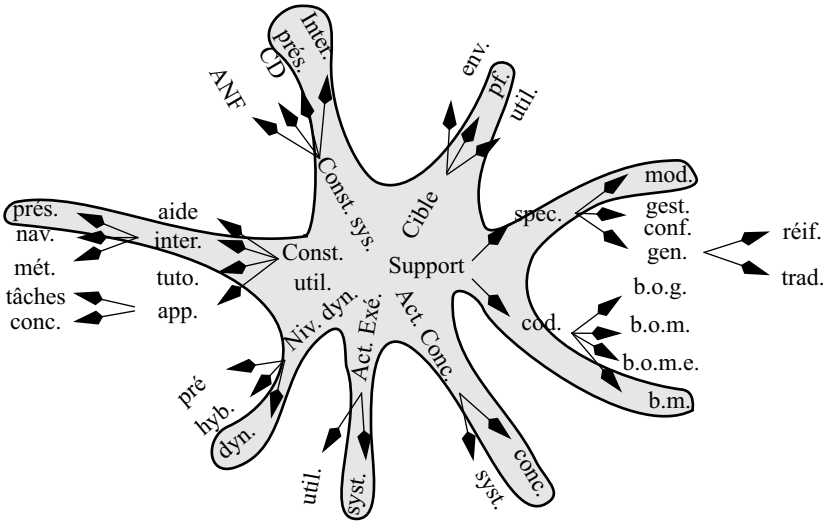
MOBI-D



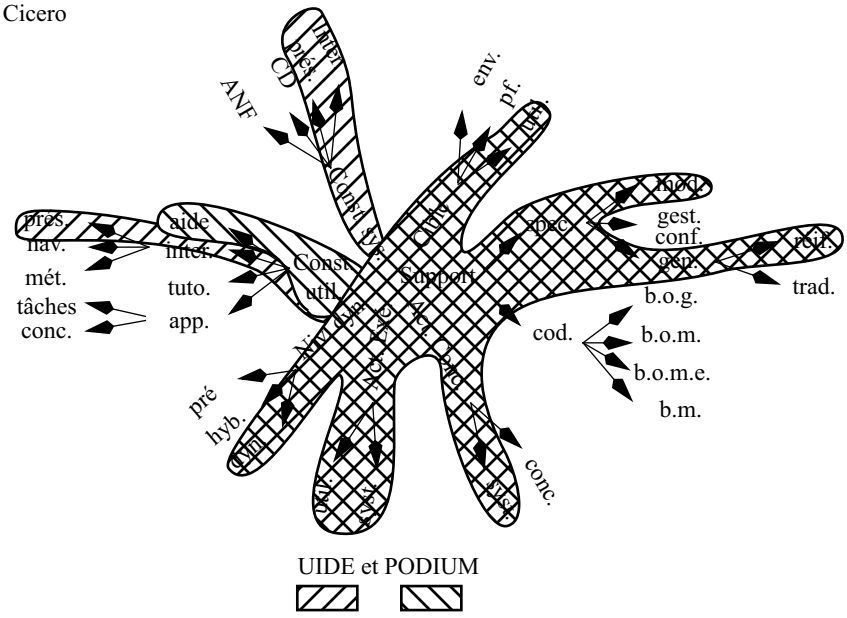
USE-IT

Figure 21. La fleur des outils pour IHM multicibles.

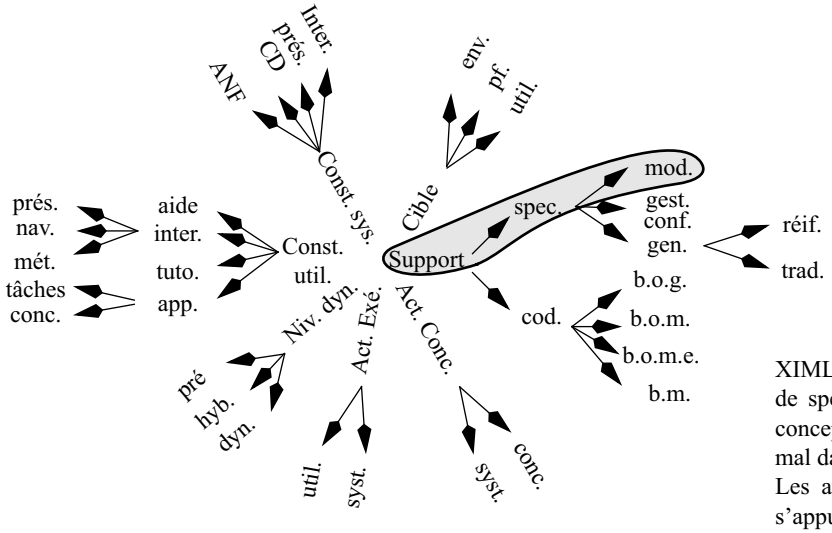




Cicero



UIDE et PODIUM



XIML

XIML se limitant aujourd'hui à un langage de spécification (pas d'environnement de conception, génération) il se projette assez mal dans notre fleur.
Les autres aspects dépendront des outils s'appuyant sur XIML.

6. Synthèse et justificatif de notre thèse

Nous venons de voir un certain nombre d'outils. Certains orientés mise en oeuvre, facilitant le développement multicontexte en proposant un langage abstrait, des mécanismes pour l'adaptation à l'exécution, une boîte à outils multiplate-forme voir multimodale. D'autres seront plutôt orientés conception et génération automatique, en proposant des systèmes de modélisation et génération. Mais ces approches ont chacune leurs limites.

6.1. POURQUOI REMONTER AU NIVEAU CONCEPTION

Une limite assez générale des approches au niveau implémentation est la non-intégration de la conception amont, et en particulier du modèle des tâches et des concepts du domaine. Comme le montrent les travaux faits en UIMS et MB-IDE, ces deux modèles ont une influence forte dans la conception et l'adaptation d'une interface. Les modèles des tâches et des concepts du domaine fourniront les informations importantes pour la réorganisation du dialogue. Cela n'enlève en rien la validité des approches de mise en oeuvre. Elles seront seulement limitées dans l'espace d'adaptation possible. Pour aller au-delà il est nécessaire de remonter dans le processus de conception.

6.2. UN MANQUE AU NIVEAU CONCEPTION

Comme nous venons de le dire, il est nécessaire de remonter au niveau de la conception pour obtenir les informations nécessaires et probablement suffisantes pour faire une adaptation qui ne se limite pas au simple changement d'interacteur, mais qui propose une adaptation du dialogue, des tâches ou des concepts. Pourtant tous les outils que nous avons vu, parlent de modélisation, de génération, mais très peu de conception muti-cible, de partage de modèles, de versionnement. Même si [Savidis et al. 2001] propose le concept de tâches polymorphiques dans l'application AVANTI [Stephanidis et al. 2001], ce système n'a pas été généralisé au reste de la conception et ne semble pas avoir été implémenté dans USE-IT.

Aussi nous estimons qu'il faut aller plus loin en offrant des outils qui aident la conception/génération multicible en intégrant les concepts de réification, de traduction et de gestion de configuration.

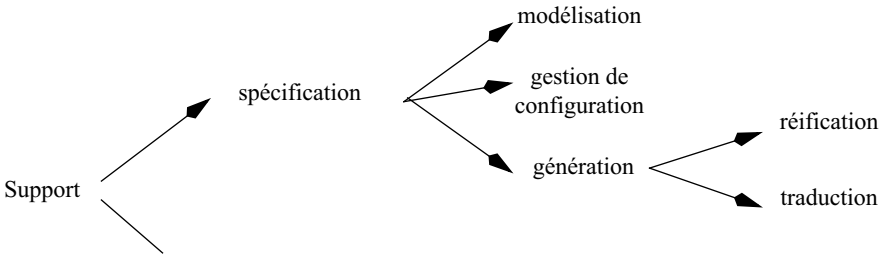
Partie 2 Cadre et Processus de référence

Chapitre 3 *Principes et Processus de référence pour la production d'IHM multicible*

Au chapitre précédent, l'analyse de l'état de l'art sur l'adaptation et notre proposition de taxonomie ont mis en évidence des lacunes en matière d'outils pour la production d'IHM multicible. Au-delà des outils de codage, qui constituent le "logiciel de base"des IHM multicibles, nous avons relevé la nécessité d'aider le concepteur dans ses activités de spécification et, par conséquent, de proposer les outils idoines. Ces outils peuvent être conçus de manière empirique ou bien se fonder sur un cadre théorique. Nous optons pour la deuxième approche.

Dans ce chapitre, nous proposons un ensemble de principes qui expriment de manière formelle les propriétés qui nous semblent fondamentales en spécification d'IHM multicible, et notamment pour les activités de modélisation, de gestion de configuration/versionnement, et de génération (cf. figure 1). La modélisation présente plusieurs avantages comme la réutilisabilité par le biais de dépôts de connaissances.

Figure 1. Support au développement. Issu du chapitre 2, figure 17.



Nous associons la notion de connaissance à celle de **description** pour laquelle nous offrons une définition formelle. Concernant la réutilisabilité, nous proposons le principe de **factorisation** qui explicite les descriptions communes à plusieurs cibles. Avec le principe de **décoration** de description, qui exprime les exceptions, nous servons à la fois la factorisation, la génération et les configuration/versionnement. Descriptions, factorisation et décoration s'inscrivent dans un **cadre théorique** qui, fondé sur la coopération des principes de **réifica-**

tion et de **traduction**, constitue un processus de référence générique et conceptuel pour la définition d'outils supports à la production d'IHM multicible.

La structure du chapitre reprend ces points comme suit :

Nous proposons en 1 une définition formelle de la notion de description, puis en 2 et 3, nous présentons les principes de factorisation et de décoration. Ces principes qui concernent la modélisation sont ensuite complétés avec la présentation des sujets relevant de la génération : les principes de réification et de traduction en 4, puis en 5 le cadre de référence dans lequel s'inscrit l'ensemble de nos principes.

1. Notion de description

En IHM, toute méthode de conception comme Diane+[Tarby 1993], PROSPECT [Brisson et Andre 94], etc., préconise la production de modèles, et notamment les modèles de tâches et des concepts du domaine. Avec la notion de description, nous désignons de manière unifiée les différentes sortes de modèles utilisées en conception d'IHM. A partir d'une notation, nous proposons en 1.1 une définition formelle de cette notion suivie en 1.2 et 1.3 de la présentation des opérations qui lui sont associées. En 1.4, nous offrons sous forme de diagramme de classes (type UML [Muller 1999]), une interprétation concrète proche de la mise en œuvre de la notion de description. Nous disposons ainsi d'un outil de raisonnement général sur lequel nous pouvons bâtir nos principes au regard des outils de spécification d'IHM multicible.

1.1. DÉFINITION ET NOTATIONS

Une description est un ensemble d'éléments liés, éventuellement, par des relations. Description et élément sont détaillés ci-dessous en séquence.

Une description

Une description D se note :

$D = \{e_1, ... , e_n\}$
où n est le cardinal de D, e_i un élément de D, et $i \in [1..n]$.

On note :

- $\mathbb{D}$, l'ensemble des descriptions constituant un dépôt de connaissances.
- D_m, la description du modèle *m*.
  Par exemple, $D_{Tâches}$ est la description du modèle des tâches. On dira aussi que $D_{Tâches}$ est le modèle de tâches. De même, $D_{Concepts}$

désigne la description du modèle des concepts.

- D_m^{Cbl}, la description du modèle *m* pour la cible *Cbl*.
  Par exemple, $D_{Tâches}^{PalmPilot}$ désigne la description du modèle des tâches pour la cible PalmPilot.

Description multicible

D_m^{Cbl} est une **description multicible** de *m* lorsque *Cbl* désigne un ensemble de cibles Cbl_1, ..., Cbl_p, soit Cbl = {Cbl_1, ..., Cbl_p}.

Un élément

Soit $E^<$ la classe abstraite élément dont l'attribut *id* sert d'identificateur unique.

$E^<$ se spécialise en la classe élément abstrait de type variable ${}_vE^<$ et en la classe élément abstrait de type relation ${}_rE^<$. ${}_rE^<$ définit une relation (au moins binaire) entre éléments. Elle possède l'attribut "Met en relation" qui est le vecteur des éléments mis en relation.

${}_vE^<$ et ${}_rE^<$ se spécialisent en une classe concrète élément E.

Par définition, un élément *e* est une instance de E. Il est de type variable ou relation.

On note :

- *Isa(e)*, l'ensemble des classes dont *e* est une instance [Ducournau 1996].
- *Attr(e)*, l'ensemble des attributs de *e* [Ducournau 1996].

La figure 6 montre une représentation équivalente en diagramme de classes, mais orientée mise en œuvre, de ces définitions. Nous les complétons maintenant par la définition d'opérations.

1.2. OPÉRATIONS SUR LES ÉLÉMENTS

Nous proposons la pseudo-égalité entre éléments et la substitution d'éléments. Ces opérations vont servir de base à la définition d'opérations sur les descriptions. A leur tour, ces opérations permettront d'étayer nos principes de factorisation et de décoration.

Pseudo-égalité

Soient e_1 et e_2 deux éléments de E. e_1 et e_2 sont pseudo-égaux[1] (noté ≅) :

1 - si ce sont des instances des mêmes classes, c'est-à-dire :

1. On parle de pseudo-égalité, puisque, par définition : $e_1.id \neq e_2.id$. Par commodité d'écriture, nous utiliserons la notation pointée pour désigner la valeur d'un attribut. Pour plus de précision, se référer à [Ducournau 1996]

$Isa\ (e_1) = Isa\ (e_2)$

2 - et si leurs attributs sont de valeurs égales ou pseudo-égales, c'est-à-dire :

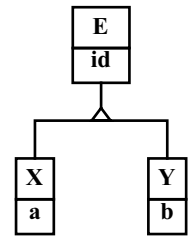
$Attr\ (e_1) = Attr\ (e_2)$ et

- $e1.a_i = e2.a_i$ pour tout attribut a_i de type simple (entier, flottant, caractère, ...)
- $e1.a_i \cong e2.a_i$ pour tout attribut a_i de type élément (cas d'un élément de type relation qui pointe sur des éléments).

La figure 2 illustre la pseudo-égalité entre éléments.

Cette définition devra être affinée en fonction des éléments considérés (par exemple : égalité entre tâches, égalité entre concepts du domaine). En particulier est-ce que deux éléments peuvent être considérés comme égaux s'ils n'ont pas les mêmes relations ? Nous ne rentrerons pas plus dans les détails de la pseudo-égalité. Pourtant il sera nécessaire de mieux la définir lors de l'implémentation puisque, comme nous le verrons par la suite, de nombreux mécanismes s'appuient sur cette relation.

Figure 2. Exemples de pseudo-égalité entre éléments. Soient les intances e_1, e_2, e_3, e_4 de X et Y, deux sous-classes de E.



définition de classes

$e_1 = X\ (a = 1, id = 1)$;
$e_2 = X\ (a = 2, id = 2)$;
$e_3 = Y\ (b = 1, id = 3)$;
$e_4 = X\ (a = 1, id = 4)$;

définition d'instances

$e_1 \not\cong e_2$ Car $e_1.a \neq e_2.a$

$e_1 \not\cong e_3$ Car $Isa\ (e_1) \neq Isa\ (e_3)$

$e_1 \cong e_4$ Car $Isa\ (e_1) = Isa\ (e_4)$, $Attr\ (e1) = Attr\ (e_4)$ et $e_1.a = e_2.a$.

Substitution d'éléments

Soient ${}_ve_1$ et ${}_ve_2$ deux éléments pseudo-égaux de type variable et ${}_re_3$ un élément de type relation telles que ${}_ve_1$ est mis en relation par ${}_re_3$ mais pas ${}_ve_2$. Alors la substitution de ${}_ve_1$ par ${}_ve_2$ dans ${}_re_3$ est la fonction S notée $S\ ({}_ve_1, {}_ve_2, {}_re_3)$ -> ${}_re_3$' qui substitue ${}_ve_1$ par ${}_ve_2$ dans ${}_re_3$ pour produire ${}_re_3$'.

Autrement dit :

$$\text{Soient } (e_1, \ldots, e_n) \text{ tels que } \forall (i \in [1 \ldots n]), {}_{v}E^{<} \in \text{Isa}(e_i)$$

$$\text{Soit } e_q \text{ tel que } {}_{r}E^{<} \in \text{Isa}(e_q) \wedge e_q = R(e_1, \ldots, e_n)$$

$$\text{Soit } e_j \text{ tel que } j \notin [1 \ldots n] \wedge {}_{v}E^{<} \in \text{Isa}(e_j) \wedge \exists (p \in [1 \ldots n]) \text{tel que } (e_p \cong e_j)$$

Alors e_p peut-être substitué par e_j dans $e_q = R\ (e_1,..., e_p,..., e_n)$ pour produire $e_q' = R\ (e_1,..., e_j,..., e_n)$.

Exemple d'une substitution dans une relation binaire : Soient ${}_{v}e_1$, ${}_{v}e_2$, ${}_{v}e_3$ trois tâches telles que ${}_{v}e_1 \cong {}_{v}e_3$, et ${}_{r}e_4$ = est_fille_de (${}_{v}e_2$, ${}_{v}e_1$) qui exprime que la tâche ${}_{v}e_2$ est fille de ${}_{v}e_1$. L'application de S (${}_{v}e_1$, ${}_{v}e_3$, ${}_{r}e_4$), substitution de ${}_{v}e_1$ par ${}_{v}e_3$ dans ${}_{r}e_4$, produit la relation ${}_{r}e_4'$= est_fille_de (${}_{v}e_2$, ${}_{v}e_3$) dans laquelle ${}_{v}e_2$ est maintenant fille de ${}_{v}e_3$.

1.3. OPÉRATIONS SUR LES DESCRIPTIONS

Nous définissons quatre opérations sur les descriptions : la pseudo-égalité, la pseudo-intersection, la pseudo-inclusion et la pseudo-union. La pseudo-intersection sert à définir formellement notre principe de factorisation tandis que la pseudo-union définit notre notion de représentation multicible de modèle introduite dans la section 1.1.

Pseudo-égalité

Deux descriptions D_1 et D_2 sont pseudo-égales (notée $\cong$), si elles sont formées d'éléments pseudo-égaux :

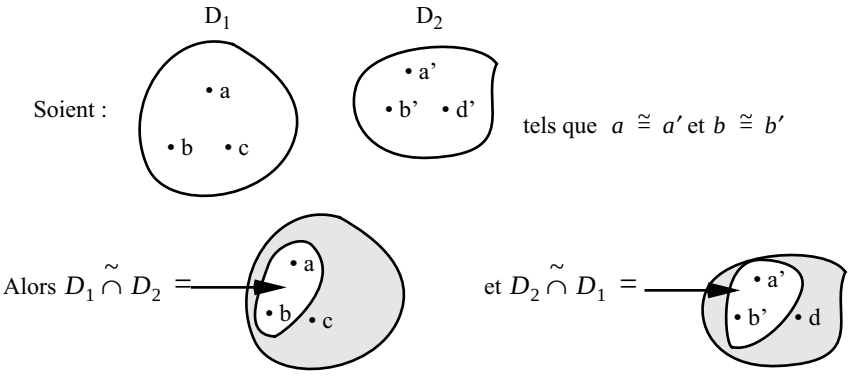
$$D_1 \cong D_2 \Leftrightarrow \begin{cases} \forall e \in D_1, \exists e' \in D_2, \text{tq} \quad e \cong e' \\ \forall e' \in D_2, \exists e \in D_1, \text{tq} \quad e' \cong e \end{cases}$$

Pseudo-intersection

La pseudo-intersection des descriptions D_1 et D_2 (notée $\tilde{\cap}$) est l'ensemble des éléments pseudo-égaux de D_1 et D_2 :

$$D_1 \tilde{\cap} D_2 = \{ e \in D_1, \exists e' \in D_2, \text{tq} \quad e \cong e' \}$$
$$D_2 \tilde{\cap} D_1 = \{ e' \in D_2, \exists e \in D_1, \text{tq} \quad e' \cong e \}$$
$$D_1 \tilde{\cap} D_2 \cong D_2 \tilde{\cap} D_1$$

Figure 3. Exemple de pseudo-intersection entre les descriptions D1 et D2.



La figure 3 montre un exemple de pseudo-intersection.

Pseudo-inclusion La description D_1 est pseudo-incluse dans D_2 (notée $\tilde{\subset}$) si l'ensemble des éléments de D_1 sont inclus dans D_2 :

$$D_1 \tilde{\subset} D_2 \Leftrightarrow \forall e \in D_1, \exists e' \in D_2, \text{tq } e \cong e'$$

Pseudo-union La pseudo-union des descriptions D_1 et D_2 (notée $\tilde{\cup}$) est l'union de D_1 et de D_2 dans laquelle les éléments de D_2 sont substitués par leurs pseudo-égaux dans D_1 (voir les exemples des figures 4 et 5).

Construction d'une pseudo-union :

Soient $D_1 \in \mathbb{D}, D_2 \in \mathbb{D}, D = D_1 \tilde{\cap} D_2, D' = D_2 \tilde{\cap} D_1$

Par définition, $\forall e_1 \in D, \exists e_2 \in D'$

Soit $D_2' = D_2 - D'$

Soit $D_2'' = \{e \in D_2' \text{ tq } {}_vE^< \in \text{Isa}(e)\}$

Soit $D_2''' = \{e \text{ tq } \forall e' \in D_2' \text{ tq } {}_vE^< \in \text{Isa}(e'), e = \text{Substitution}(e_2, e_1, e')\}$

soit $D_1 \tilde{\cup} D_2 \overset{\text{def}}{=} D_1 + D_2'' + D_2'''$. Nous avons $D_1 \tilde{\cup} D_2 \cong D_2 \tilde{\cup} D_1$.

Figure 4. Exemple de pseudo-union entre deux descriptions : cas simple.

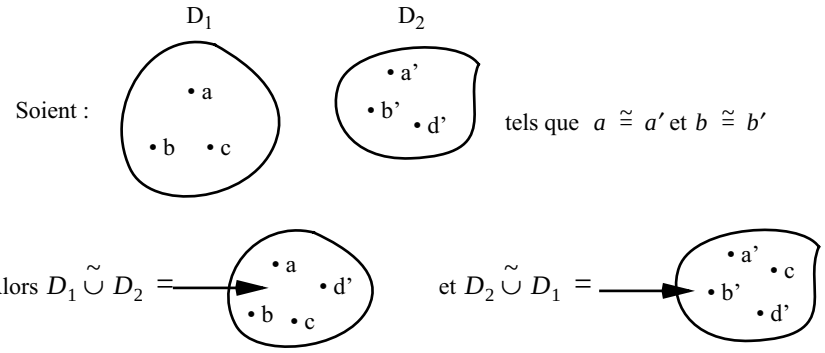
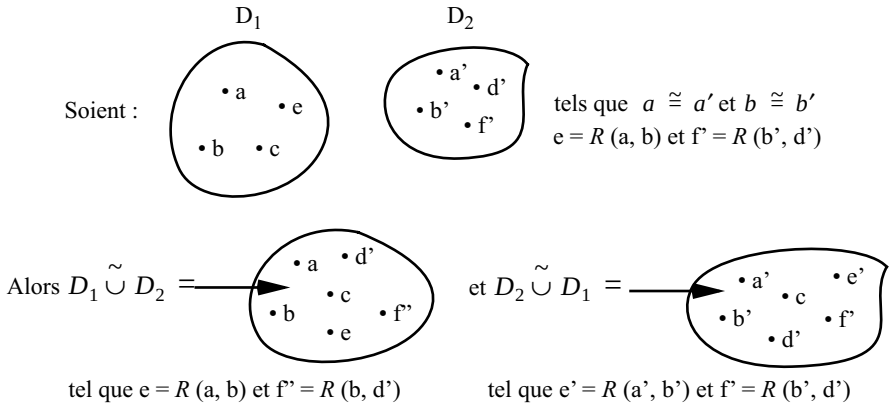


Figure 5. Exemple de pseudo-union entre deux descriptions : cas général. Il y a substitution dans *e* et *f'* pour produire respectivement *e'* et *f''*.



Description multicible

Soit Cbl un ensemble de cibles : Cbl = {Cbl_1, ..., Cbl_p} et D_m^{Cbl1}, ..., D_m^{Cblp}, les descriptions du modèle m pour les cibles Cbl_1, ..., Cbl_p respectivement. La description multicible D_m^{Cbl} de m est la pseudo-union des descriptions monocibles, soit :

$$D_m^{Cbl} = D_m^{\{Cbl_i, ..., Cbl_i\}} = \tilde{\bigcup_{\forall Cbl_i \in Cbl}} D_m^{Cbl_i}$$

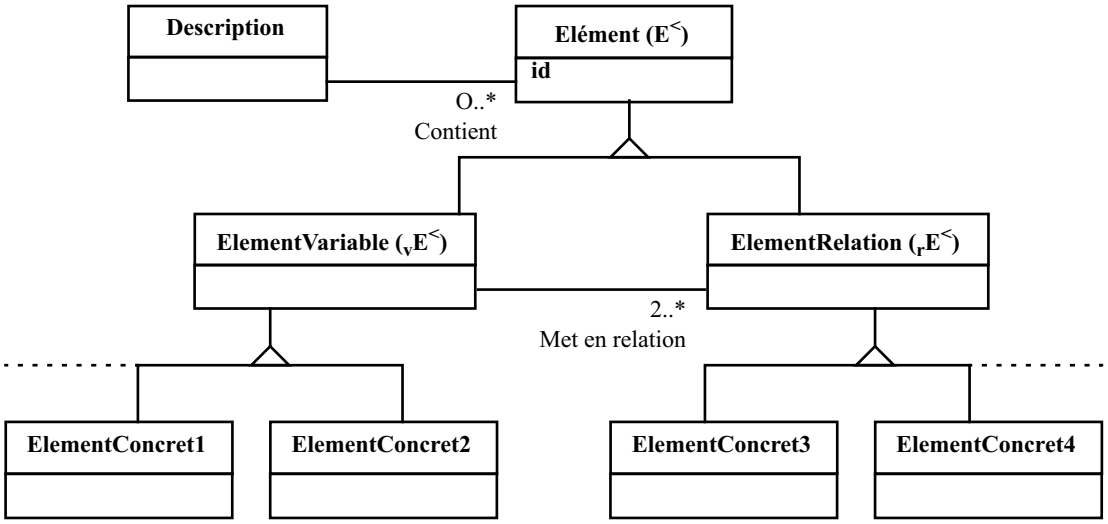
1.4. INTERPRÉTATION APPLIQUÉE

Le diagramme de classes de la figure 6 montre une représentation orientée implémentation des notions introduites jusqu'ici :

- l'Elément. C'est une classe abstraite qui représente un élément atomique de description ;
- l'ElémentVariable qui est un Elément de type variable ;
- l'ElémentRelation qui est un Elément de type relation ;
- l'ElémentConcret. C'est la concrétisation d'un ElémentVariable ou Relation. Par exemple, pour la description d'un arbre de tâches, cette classe peut-être l'élément variable Tâche et l'élément relation OpérateurEnchaînementTâches ;
- la Description est la classe principale. C'est un ensemble d'éléments abstraits.

Nous proposons ici une spécialisation de Elément en deux classes Variable et Relation. Mais cette information peut-être codée sous la forme d'un attribut dans Elément.

Figure 6. Application de la description. Diagramme de classes montrant une implémentation possible de notre notion de description.



Pour un métier donné, les descriptions, qui varient peu d'une application à l'autre, constituent des dépôts de connaissances qu'il convient de réutiliser. En IHM multicible, la réutilisabilité s'impose de facto : aux

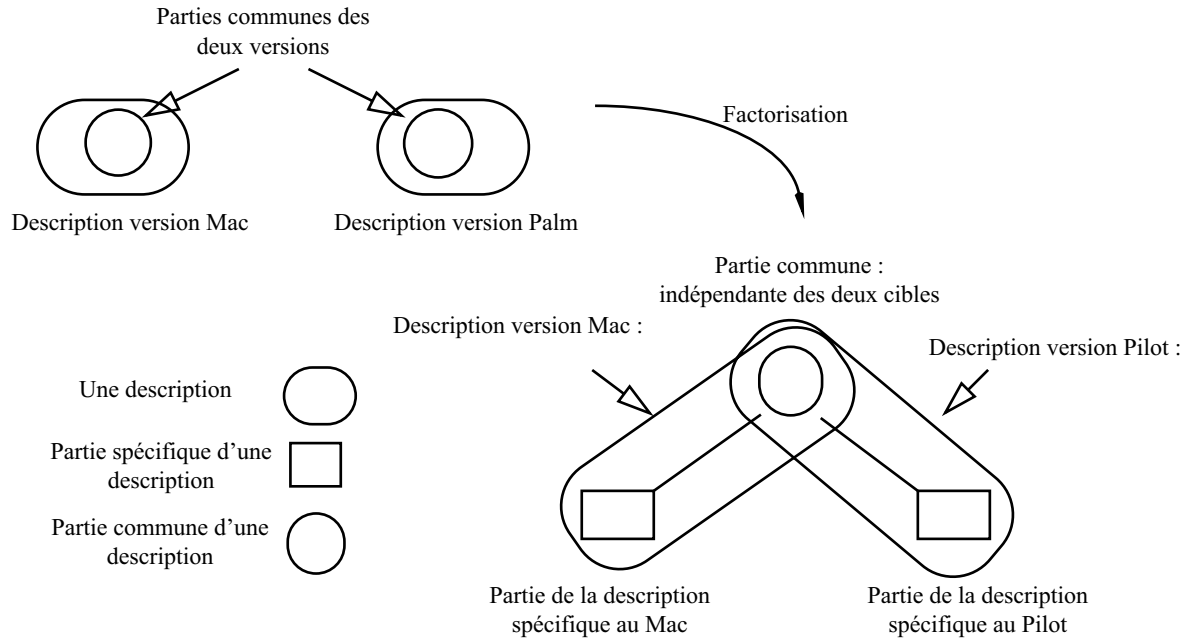
cibles qui partagent certaines caractéristiques, doivent correspondre des descriptions communes réutilisables. Le principe de factorisation répond à ce besoin.

2. *Principe de factorisation*

La factorisation, à la manière de la généralisation/spécialisation en programmation par objets, permet de distinguer pour une description donnée, les parties communes à plusieurs cibles donc indépendantes des cibles, des parties spécifiques dépendantes des cibles. On appelle **divergence** l'apparition de parties spécifiques dans une description.

La figure 7 montre, de manière intuitive, l'effet de la factorisation.

Figure 7. Le principe de factorisation. Deux descriptions partagent une partie commune et possèdent chacune une partie spécifique.



2.1. DÉFINITION ET NOTATIONS

Soient :

- $Cbl = \{Cbl_1, ..., Cbl_p\}$, $\forall i \in [1 \ldots p]\ D_m^{Cbl_i}$, et $D_m^{Cbl} = \tilde{\bigcup}_{\forall Cbl_i \in Cbl} D_m^{Cbl_i}$
- ${}_cD_m{}^{Cbl}$, la partie commune de $D_m{}^{Cbl}$.

La factorisation appliquée à un ensemble de descriptions $D_m^{Cbl_i}$ se définit comme la pseudo-intersection de ces descriptions :

$$_cD_m^{Cbl} = \underset{\forall Cbl_i \in Cbl}{\tilde{\bigcap}} D_m^{Cbl_i}$$

Soit $Cbl_i \in Cbl$.

La partie spécifique de D_m^{Cbl} correspondant à la cible Cbl_i est notée $_{Cbli}D_m^{Cbl}$. Elle se définit comme la description D_m^{Cbl} moins sa partie commune $_cD_m^{Cbl}$:

$$_{Cbl_i}D_m^{Cbl} = D_m^{Cbl_i} - {}_cD_m^{Cbl} = D_m^{Cbl_i} \tilde{\cap} {}_cD_m^{Cbl}$$

Il en résulte que :

$$D_m^{Cbl_i} = {}_{Cbl_i}D_m^{Cbl} \tilde{\cup} {}_cD_m^{Cbl}$$

Par exemple, soient :

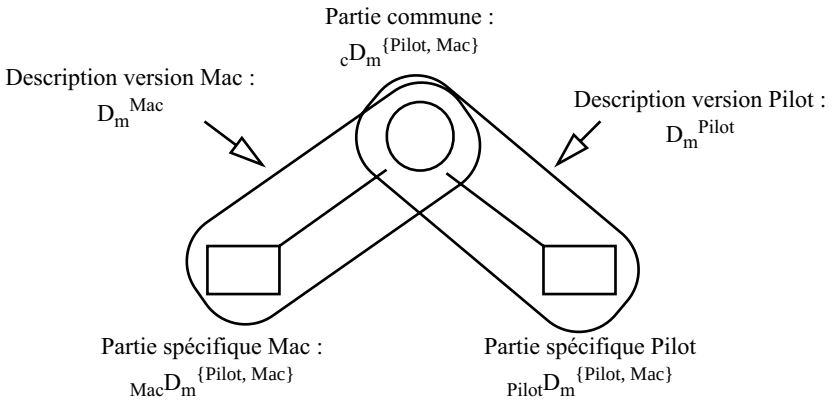
- $D_{Taches}^{\{Pilot, Mac\}}$, la description des tâches pour les cibles PalmPilot et Macintosh,
- D_{Taches}^{Pilot}, la description des tâches pour le Pilot seulement.

Alors, D_{Taches}^{Pilot} se factorise en :

$$D_{\text{Tâches}}^{\text{Pilot}} = {}_cD_{\text{Tâches}}^{\{\text{Pilot, Mac}\}} \tilde{\cup} {}_{\text{Pilot}}D_{\text{Tâches}}^{\{\text{Pilot, Mac}\}}$$

La figure 8 illustre sous forme graphique l'effet de la factorisation.

Figure 8. Effet de la factorisation. La description $Dm^{\{Pilot, Mac\}}$ en une partie commune et en deux parties spécifiques pour le Pilot et le Mac respectivement.



2.2. INTERPRÉTATION APPLIQUÉE DU PRINCIPE

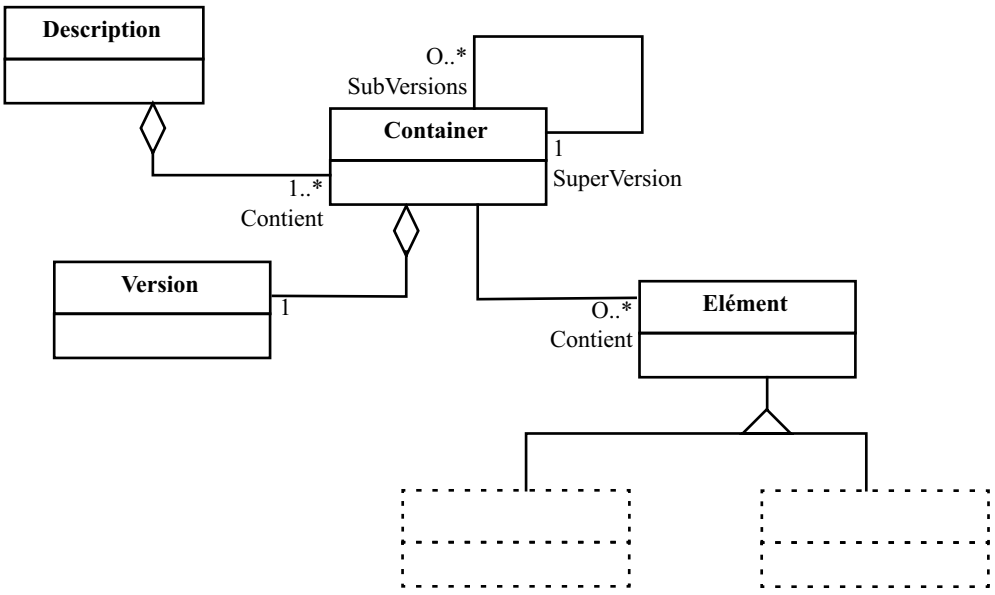
En pratique, le principe de factorisation est à rapprocher du concept de configuration/versionnement du Génie Logiciel. En effet, une description, avec ses parties communes et spécifiques, peut se voir comme un composant participant à de multiples configurations et existant sous de multiples versions. La figure 9 présente une interprétation révisée de notre notion initiale de description de la figure 6.

Le diagramme proposé définit deux classes supplémentaires au diagramme initial :

- la Version. Cette classe décrit toutes les informations nécessaires à l'identification d'une version de description. Par exemple un identificateur, le nom ou les noms des créateurs de la version, la date de création, etc. ;
- le Container. Cette classe représente une nouvelle version de la description. Elle représente un ensemble d'éléments (liens d'associations vers la classe abstraite Elément) et est estampillée d'une version (lien d'agrégation avec la classe Version).

La Description est maintenant composée d'un ou plusieurs containers (lien d'agrégation multiple vers la classe Container).

Figure 9. Diagramme de classes d'une description multiversion.



Ce diagramme décrit un patron qui peut-être implémenté selon différentes sémantiques. Une sémantique possible est la suivante :

- Soit une version C1 sur laquelle un concepteur veut faire un correctif de bogues. Alors dans un nouveau container BugFix1 qui est SubVersion de C1, sont sauvées les différences avec C1.
- Si maintenant le concepteur veut créer une nouvelle version

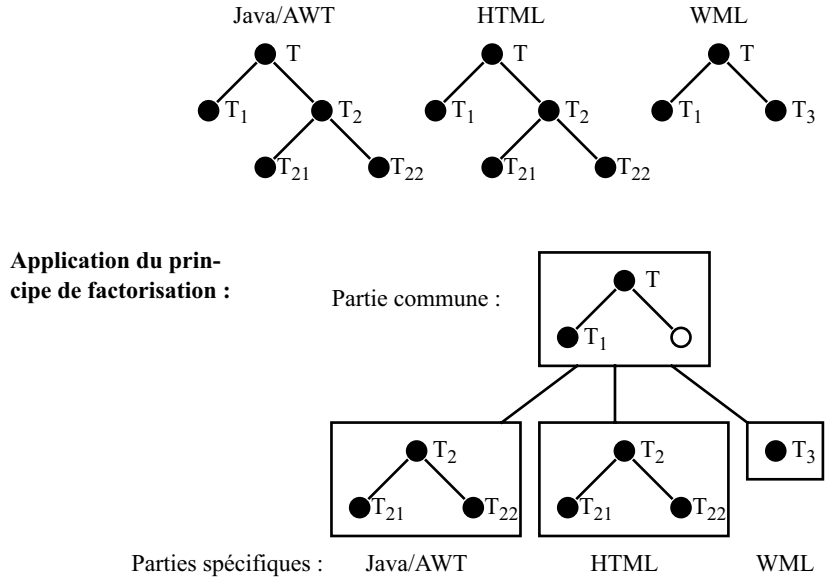
complète C2 = C1 + BugFix1, alors il crée un nouveau container C2 contenu dans Description.

Pour la description d'un modèle pour une cible Cbl, nous appliquons le diagramme avec la sémantique suivante :

- Une description contient un unique container C;
- C est la partie commune de la description. Il correspond à ${}_{c}D_{m}^{Cbl}$;
- C contient éventuellement plusieurs containers C_i qui correspondent aux parties spécifiques ${}_{Cbli}D_{m}^{Cbl}$ (où $Cbl_i \in Cbl$);
- Un container C_i ne peut pas avoir de sous-version de container ;
- Une version contient au moins un identificateur de Cbl.

La figure 10 montre un exemple de factorisation appliquée à une description de modèle de tâche. Cet exemple définit de manière formelle le concept de tâches polymorphiques de AVANTI [Stephanidis et al. 2001]. En proposant le principe de factorisation, nous généralisons à tout type de description (tâches, concept du domaine, etc.) le polymorphisme que Stéphanidis a restreint aux arbres de tâches.

Figure 10. Exemple de factorisation. Trois arbres de tâches pour différentes plates-formes.



Les parties communes et spécifiques de descriptions peuvent être spécifiées par le concepteur. Dans ce cas, l'opération de factorisation est réalisée par le concepteur. Ou bien, les parties communes et spécifiques de descriptions sont calculées par un outil. Dans ce cas, l'outil doit avoir accès à un type de connaissance pour remplir cette fonction. Notre principe de décoration, qui permet l'expression des exceptions, vise cet objectif.

3. Principe de décoration

Au sens général, une décoration est l'action de décorer. Elle désigne aussi l'objet qui joue le rôle d'ornement. Nous reprenons cette double perspective appliquée aux éléments d'une description.

Définition. **La décoration** en tant qu'action est l'opération d'ajout d'une décoration à un élément de description.

Définition. **Une décoration** en tant qu'ornement est une information associée à un élément de description. Cette information, parce qu'elle est externe à l'élément, ne modifie pas la description.

Dans le cadre de notre étude, une décoration peut servir un ou plusieurs objectifs :

- La modification de description : c'est la décoration corrective ;
- La factorisation de description : c'est la décoration de factorisation ;
- L'expression de directive de génération : c'est la décoration directive.

Dans cette section, nous proposons en 3.1 une définition formelle de la notion de décoration suivie en 3.2 d'un affinement de nos diagrammes de classes orientés implémentation. Puis nous passons en revue les trois types de décoration : la décoration corrective (3.3), la décoration de factorisation (3.4) et la décoration directive (3.5). En fin de section, nous proposons une analyse de cette notion (3.6, 3.7) avec ses apports et les problèmes posés.

3.1. DÉFINITION ET NOTATION

Soient,

- ${}_{d}E^{<}$ la classe décoration
- ${}_{rd}E^{<}$ la classe définissant la relation binaire de décoration
- D_m^{Cbl} la description du modèle *m* pour la cible *Cbl*
- *e* l'élément tel que $e \in D_m^{Cbl}$
- *rd*, une instance de ${}_{rd}E^{<}$

Définition de la décoration

Une décoration, notée *d*, est une instance de ${}_{d}E^{<}$. Elle est mise en relation de décoration avec l'élément *e* par *rd* : *d rd e* que nous notons ${}^{d}e$.

Par définition, *e* ne peut-être décoré que par une décoration au plus : *rd* est injective.

Puisque *d* est un élément, il est possible de décorer une décoration. Ainsi, nous pouvons avoir $^{d2,d1}e$. Dans ce cas, soit d_2 décore d_1, soit d_2 décore *e* décoré par d_1. Ce moyen permet d'appliquer une liste de décoration sur un élément (ex : $^{dn,...,d1}e$).

L'introduction de la notion de décoration appelle une révision de la définition de la pseudo-égalité.

Redéfinition de la pseudo-égalité d'éléments

Soient e_1 et e_2 deux éléments. Ils sont dits pseudo-égaux s'ils sont pseudo-égaux (selon l'ancienne définition) et si leurs décorations, lorsqu'elles existent, sont pseudo-égales (selon l'ancienne définition), ce qui donne :

IspseudoEgale (e_1, e_2)
si pas (d_1 et d_2) tel que d_1 *rd* e_1 et d_2 *rd* e_2
alors retour ($e_1 \cong e_2$)
sinon si (d_1 et d_2) tel que d_1 *rd* e_1 et d_2 *rd* e_2
alors retour (($e_1 \cong e_2$) et IsPseudoEgale (d_1, d_2));
sinon retour FAUX;

3.2. APPLICATION DU PRINCIPE DE DECORATION

La figure 12 présente une solution d'implémentation des décorations inspirée du patron *Decorator* de [Gamma et al. 1995]. Comme le montre la figure 11, Gamma et al. relient un *Decorator* et un *Component* par un lien d'agrégation. Ce choix est influencé par le contexte d'application, les IHM graphiques, dirigé par des structures de composants fortement imbriqués. Or une agrégation est une association forte de type maître-esclave [Muller 1999] entre deux objets. Par exemple, le lien entre l'objet voiture et l'objet roue est une agrégation. Dans notre cas, nous ne voulons pas décrire une telle structure d'imbrication. Nous avons donc choisi un lien faible de type association.

Figure 11. Exemple de décoration. Tiré de [Gamma et al. 1995] pp. 175,176. aBorderDecorator <- aScrollDecorator <- aTextView, reflète la relation d'appartenance entre composants graphiques

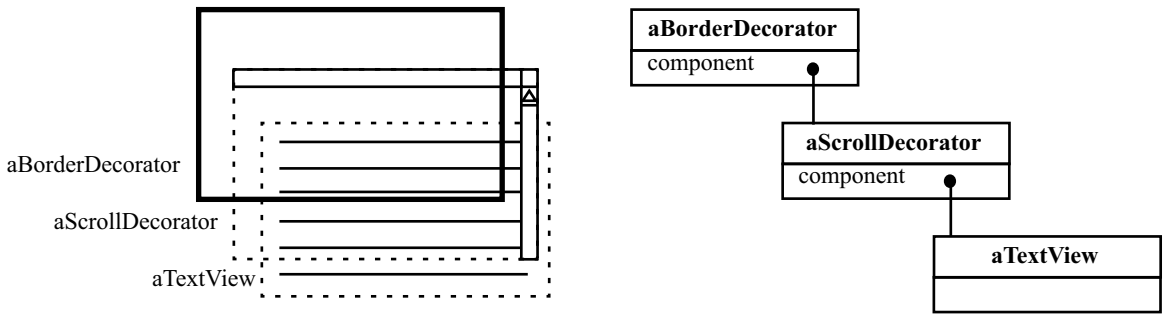
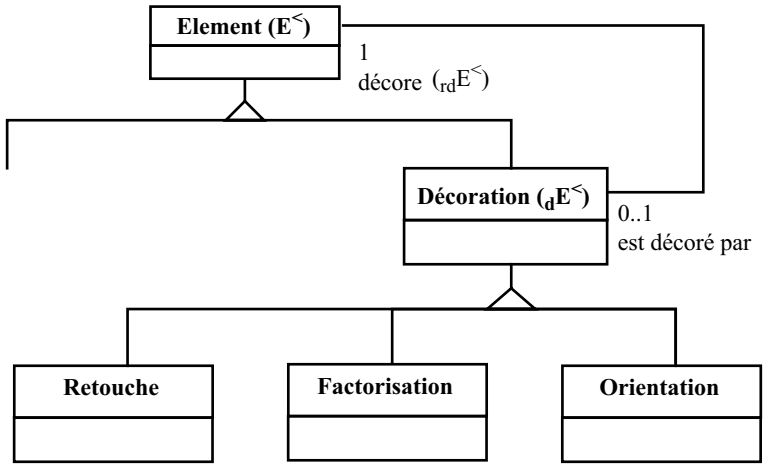


Figure 12. Diagramme de classes pour l'implémentation des décorations.



De manière plus concrète, voici des exemples de décoration d'élément :

- Pour un élément tâche : décoration qui exprime si la tâche doit être implémentée ou pas, type de la tâche, nombre de tâches filles ;
- Pour un élément concept : informations pertinentes à représenter en fonction de la cible ;
- Pour l'élément tâche/concept : pondération des liens entre la tâche et ce concept (par exemple, ce concept, pour cette tâche, est-il de première classe?);
- Pour un élément espace de travail : enchaînement avec d'autres espaces;
- Pour un élément interacteur : position, taille, etc.

Ayant introduit le concept de décoration et son modèle d'implémentation, nous décrivons maintenant nos trois types de décoration. Dans les trois cas, une décoration permet au concepteur d'exprimer une exception sans pour autant créer une nouvelle description spécifique. Minimiser les descriptions spécifiques, c'est augmenter la réutilisabilité. Chaque type d'exception répond à une finalité distincte :

- des retouches de description qui visent à corriger une erreur de modélisation,
- des exceptions au cas général pour une cible particulière,
- des indications de guidage servant les fonctions de génération.

3.3. DÉCORATION CORRECTIVE

Dans un processus itératif de développement, une décoration corrective offre au concepteur la possibilité d'ajuster une description par modifications successives et de conserver ces modifications. Parce qu'il s'agit de décoration, la description originale n'est pas modifiée directement.

Les retouches sont mémorisées à part. Ainsi, elles peuvent être réappliquées par l'outil à l'itération suivante.

La figure 13 montre l'effet d'une modification de description en l'absence de décoration. A l'itération 1 du processus de développement, le concepteur surcharge l'interface concrète générée par l'outil en agissant directement sur la description de l'IHM concrète (sur Mac, l'écran est assez grand pour permettre l'affichage de tous les concepts du domaine : Bed room, Bathroom, Livingroom). A l'itération 2 du processus, le modèle des concepts a changé (par exemple, on a décidé de ne plus exporter le concept de livingroom). L'outil de génération d'IHM concrète, qui repart d'une nouvelle description des concepts du domaine, ne peut avoir connaissance des modifications pratiquées par le concepteur à l'itération 1 sur la description de l'interface concrète du Mac.

La figure 14 montre le même exemple avec l'utilisation d'une décoration corrective. Ici, la description de l'IHM concrète n'est pas modifiée mais est décorée. A l'itération suivante, l'outil de génération, qui a accès à la décoration, produit l'effet attendu.

3.4. DÉCORATION DE FACTORISATION

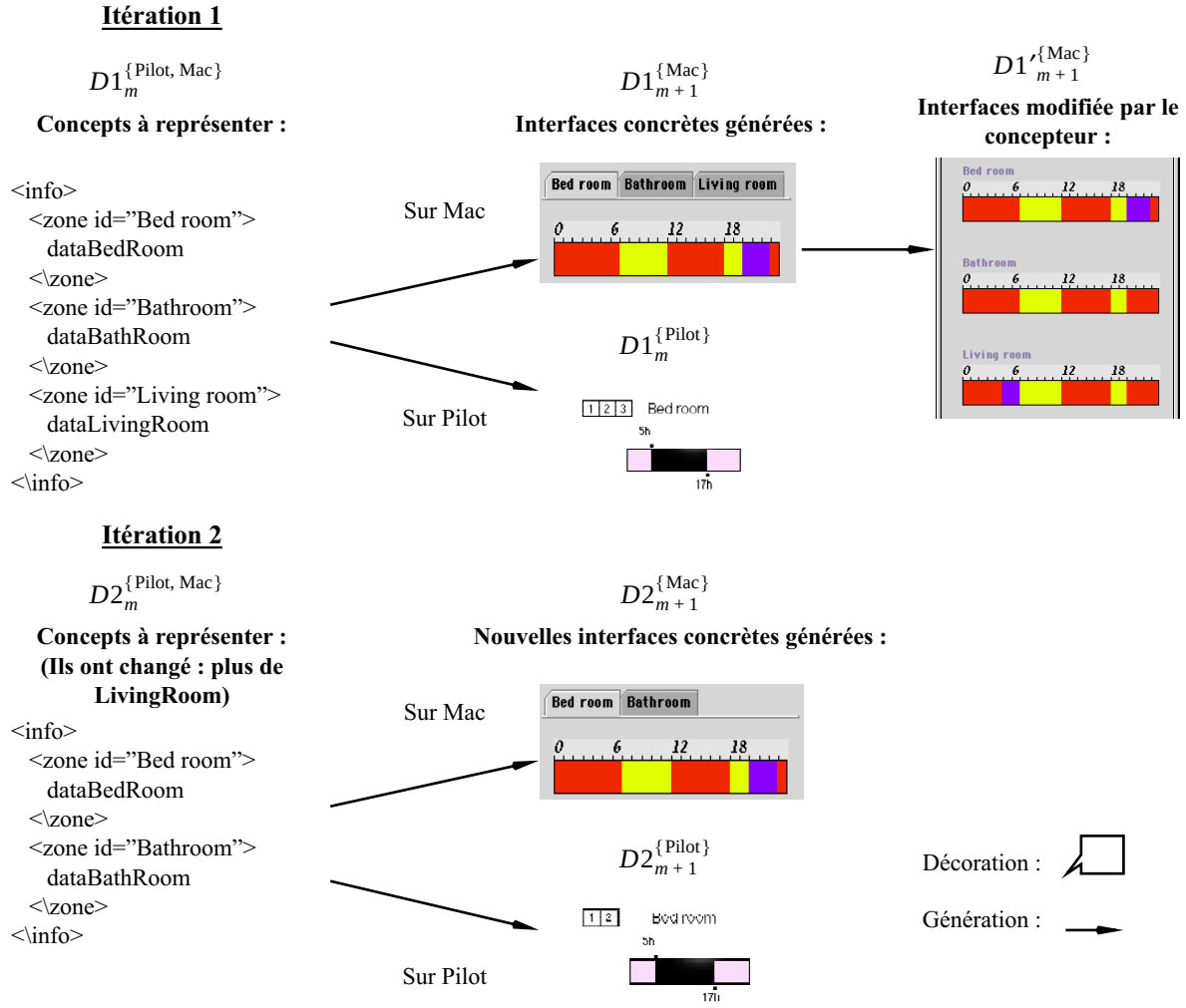
Une décoration de factorisation offre au concepteur la possibilité de spécifier une description composée d'une partie commune seulement, les spécificités étant exprimées par des annotations locales. Autrement dit, l'information dépendante de la cible n'est pas modélisée dans la description, mais décore la description.

Prenons l'exemple d'un concepteur qui souhaite développer un agenda pour PalmPilot et station Mac. L'analyse de l'activité conduit à la décision de n'implémenter sur le PalmPilot qu'un sous-ensemble des tâches prévues pour le Mac. Pour traduire cette différence, le concepteur peut procéder comme suit :

- Une première solution consiste à appliquer le principe de factorisation. Alors, le concepteur produit une description de tâches composée d'une partie commune et de deux parties spécifiques. Avec ce procédé, il peut y avoir divergence dès le départ des arbres de tâches.
- Une seconde façon consiste à utiliser une décoration de factorisation qui spécifie que telles tâches, ou tels sous-arbres de tâches, ne doivent pas être implémentés sur Palm. Ce faisant, le concepteur conserve une description de tâches sans parties spécifiques.

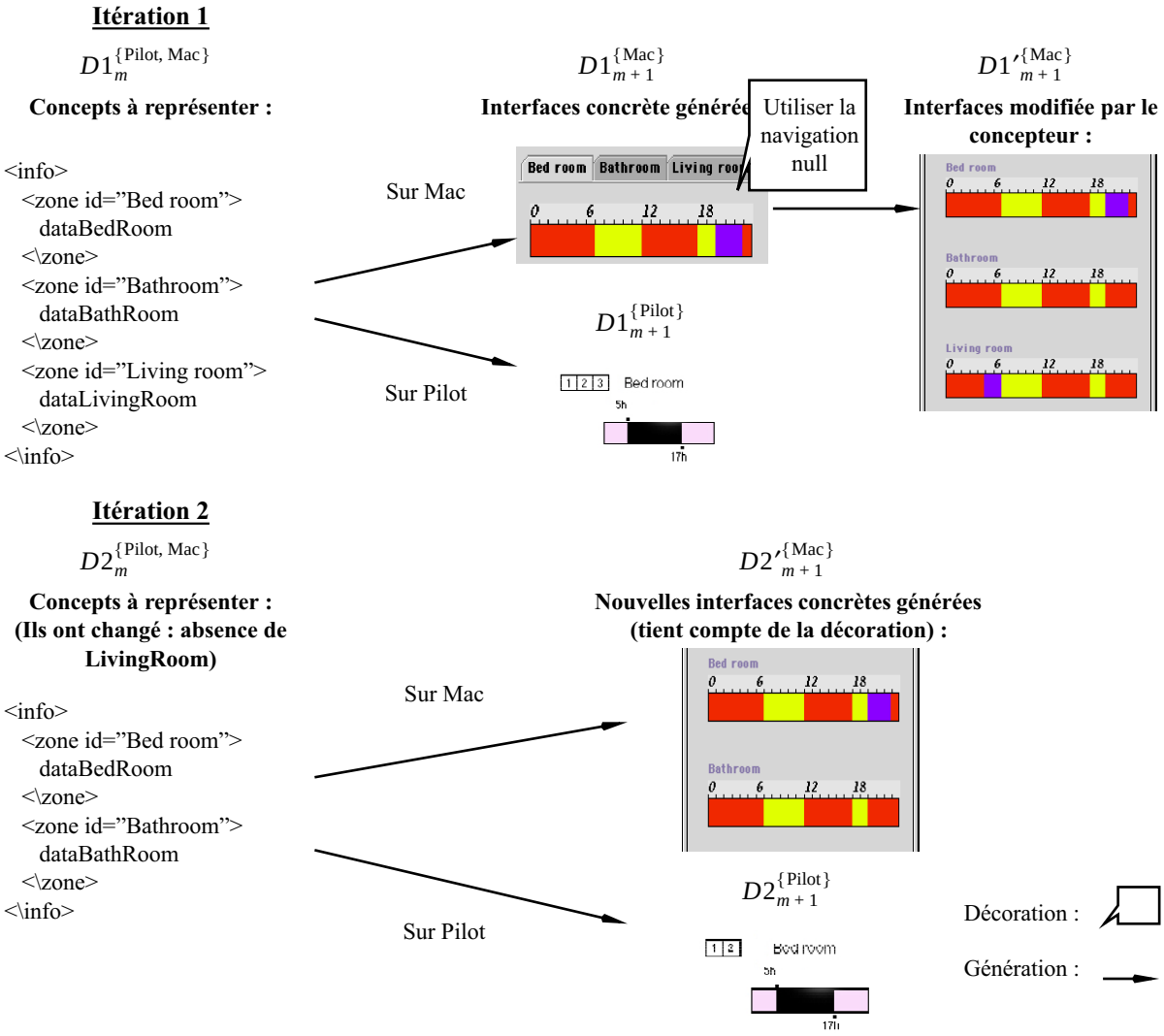
La figure 15 illustre les avantages de la décoration de factorisation pour une description à destination de trois types de plate-forme (Java/AWT, HTML et WML). Considérons une description, par exemple un modèle

Figure 13. Exemple de génération sans décoration corrective.
Itération 1 : Les interfaces concrètes $D1_{m+1}^{\{Mac\}}$ et $D1_{m+1}^{\{Pilot\}}$ sont générées par l'outil à partir de la représentation des concepts du domaine ($D1_m^{\{pilot, Mac\}}$). Pour les deux cibles, les concepts sont accessibles à l'aide d'une tâche de navigation réalisable avec des "onglets". Sur Mac, le concepteur modifie la description de l'interface concrète $D1_{m+1}^{\{Mac\}}$ pour utiliser la navigation "null" (c.-à-d. pas de navigation). Ce faisant, il produit la decription $D1'_{m+1}{}^{\{Mac\}}$.
Itération 2 : La description ($D1_m^{\{pilot, Mac\}}$) des concepts est modifiée par le concepteur en une nouvelle description ($D2_m^{\{Pilot, Mac\}}$). L'outil qui génère les descriptions des interfaces concrètes à partir de ($D2_m^{\{Pilot, Mac\}}$), a perdu la modification du concepteur sur $D1_{m+1}^{\{Mac\}}$ et regénère une navigation par onglet.



de tâches. On note que cette description est différente pour la plate-forme WML mais identique pour les deux autres. Par le principe de factorisation, nous obtenons une description commune et trois parties spécifiques à raison d'une par cible. Ainsi toute modification sur la partie spécifique de Java/AWT doit être répercutée sur la partie HTML afin de conserver leur pseudo-égalité. En posant une décoration sur le nœud à partir duquel il y a divergence pour WML, la description ne comprend qu'une partie commune. Cette décoration exprime que pour WML, "il faut faire comme ça". A partir d'une description unique,

Figure 14. Exemple de décoration corrective.
Itération 1 : comme dans l'exemple figure 13, le concepteur modifie l'interface générée mais en posant une décoration corrective au lieu de modifier directement la description.
Itération 2 : L'interface est regénérée en mais en tenant compte de la modification faite dans la première itération.



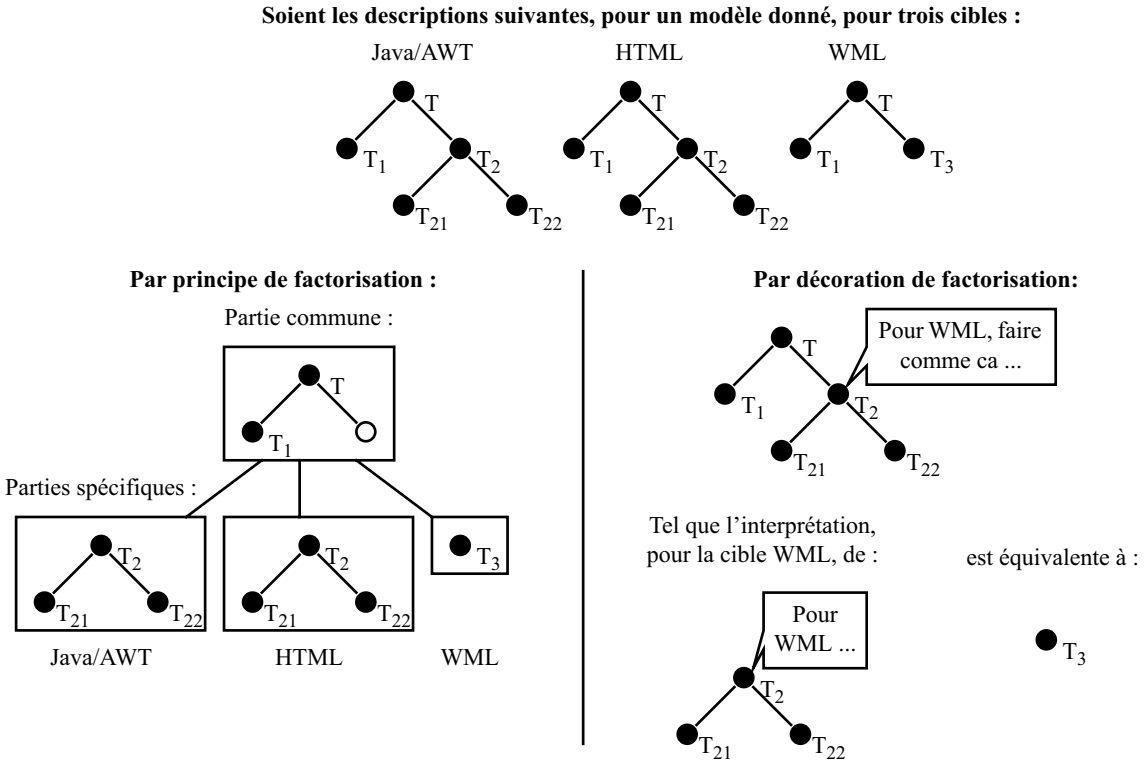
l'interprétation de la décoration de factorisation par l'outil produira la description adéquate pour WML (et de même pour les deux autres cibles).

3.5. LA DÉCORATION DIRECTIVE

Une décoration directive permet de guider le processus de génération pour traiter un cas particulier qui ne peut être formalisé sous forme d'une règle de génération générale.

Soit, par exemple, un outil de génération qui produit une description d'IHM concrète à partir d'une description des concepts du domaine. Cet outil dispose d'une règle générale qui stipule qu'un entier doit toujours être représenté par un "Label". Toutefois, le concepteur souhaite qu'un entier particulier doit être représenté au moyen d'une jauge ou

Figure 15. Avantage de la décoration de factorisation.



d'un objet artistique non standard et donc difficilement exprimable par une règle. L'utilisation d'une décoration directive répond à la situation.

La figure 16 montre un exemple d'utilisation de décoration directive. Par défaut, les règles d'inférence de l'outil de génération d'IHM concrète à partir d'une description des concepts du domaine, proposent une navigation par "onglets". Ici, Le concepteur décore la description des concepts du domaine avec une directive de génération stipulant que pour la production de la description d'IHM concrète du Mac, il ne faut pas de navigation (navigation null).

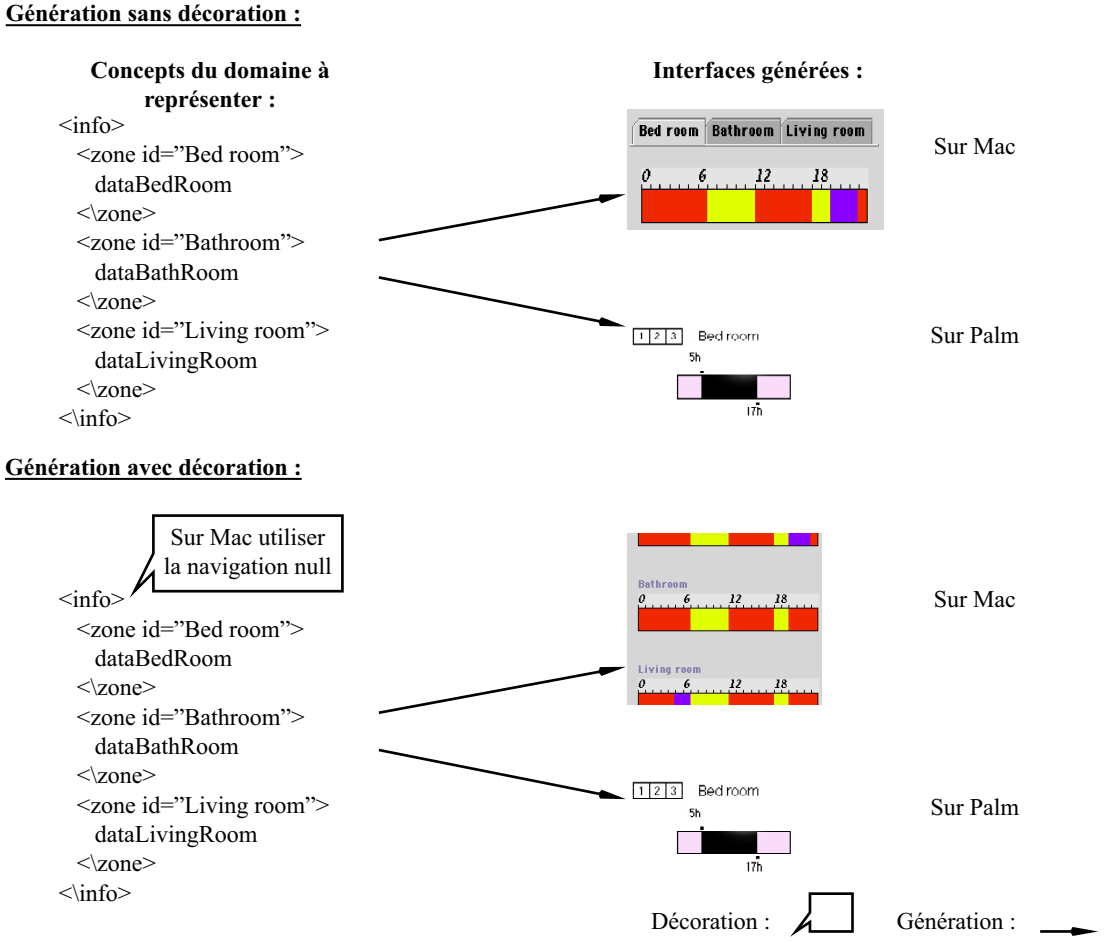
Ayant présenté les trois finalités d'une décoration, analysons de plus près sa consommation par un outil.

3.6. CONSOMMATION DE DÉCORATION

Définition. **La consommation** d'une décoration d'un élément de description est l'action de produire une nouvelle description en utilisant l'information codée dans la décoration.

La figure 17 illustre le principe de consommation d'une décoration. En haut à gauche, une description d'IHM concrète telle qu'elle est modélisée en interne. Celle-ci n'est pas décorée. En haut au centre, la même description, mais dotée d'une décoration corrective. La consommation de cette décoration produit une nouvelle description interne (en haut à droite). La décoration est désormais perdue puisqu'elle a été consom-

Figure 16. Exemple de décoration directive. Les interfaces concrètes pour Macintosh et PalmPilot sont générées à partir de la description des concepts du domaine. Pour les deux cibles, les données sont accessibles à l'aide d'une tâche de navigation réalisable avec des "onglets". Le concepteur pose une décoration pour diriger la génération sur Mac : je veux la navigation "null" (c.-à-d. absence de navigation).



mée. Les trois illustrations du bas montrent la vue externe qu'un outil (comme ARTStudio) pourrait présenter au concepteur. En l'absence de décoration, la vue externe de l'IHM concrète (en bas à gauche) offre une navigation par onglet (production selon les règles générales). Avec une décoration interprétée mais non consommée, la vue externe (en bas au centre) montre l'ensemble des concepts du domaine que l'on obtiendrait si l'on consommait la décoration. Bien sûr, cette vue est identique à celle du bas à droite qui correspond au cas de la décoration consommée. En somme, l'interprétation de décoration, c'est "pour voir", alors que la consommation est une transformation en dur.

La figure 18 montre les effets de la consommation d'une décoration d'un élément selon que cet élément appartient à une description commune ou à une description spécifique :

- La consommation d'une décoration d'élément de description commune peut avoir un effet sur la partie commune seulement (cas 1a), ou bien avoir des effets de bord à la fois sur la partie commune et les

parties spécifiques (cas 1b), voire entraîner la création d'une nouvelle partie spécifique (cas 1c).

- La consommation d'une décoration d'élément de partie spécifique a un impact sur cette partie seulement (cas 2a) mais peut entraîner la production d'une nouvelle partie spécifique (cas 2b).

Figure 17. Interprétation et consommation d'une description.

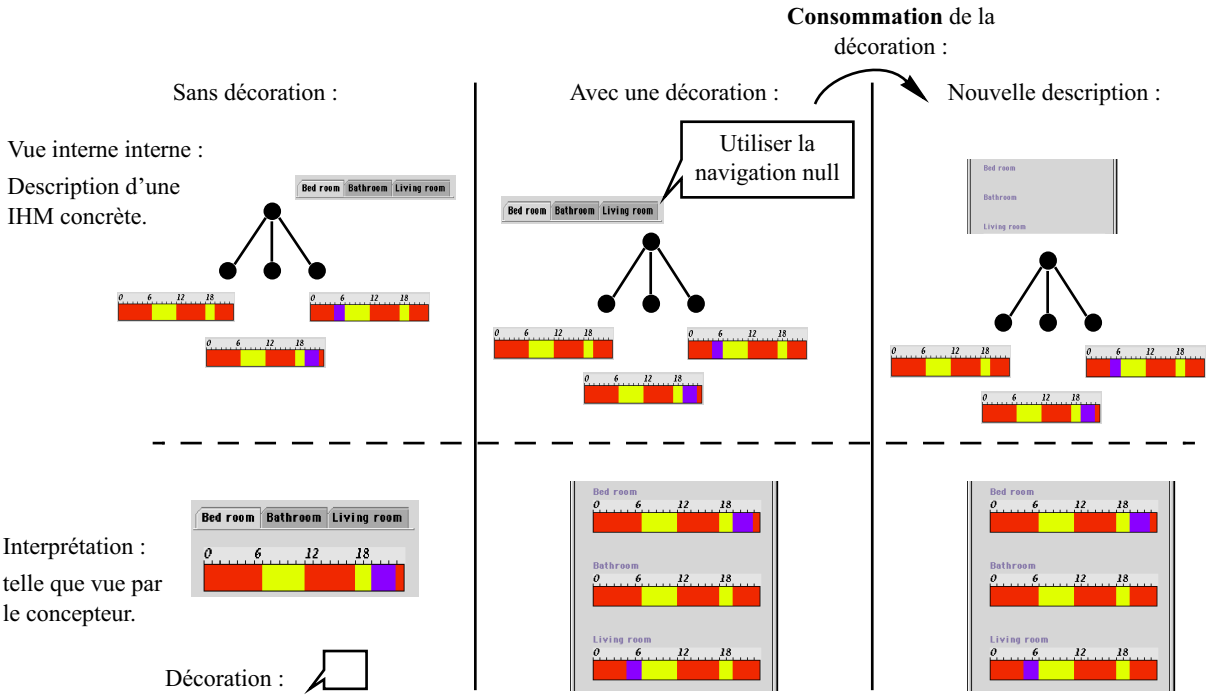
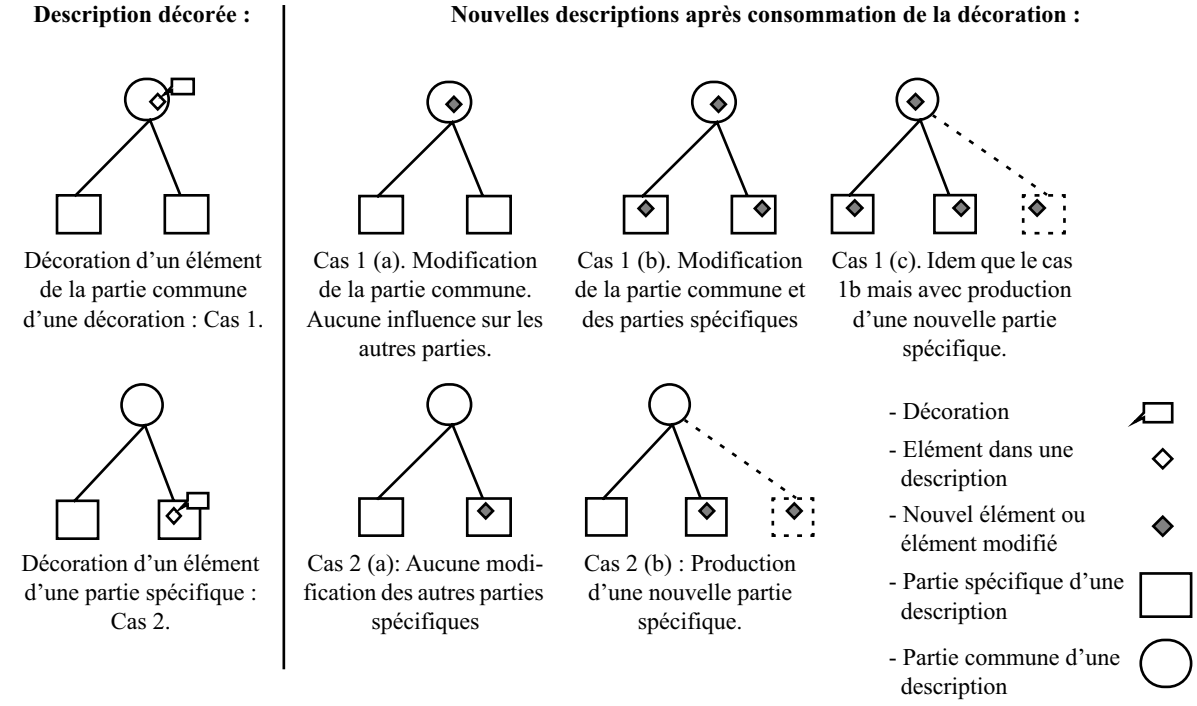


Figure 18. Consommation d'une décoration : effets possibles sur les descriptions.



3.7. ANALYSE Le principe de décoration offre plusieurs avantages. En particulier, une décoration permet de :

- différer les divergences facilitant ainsi le maintien de la cohérence entre descriptions multicibles ;
- regrouper les exceptions facilitant ainsi le repérage des différences entre cibles ;
- modifier les descriptions par "touche successive" sans perdre les modifications au cours d'un processus itératif de développement ;
- surcharger les directives de génération, offrant ainsi une personnalisation mais aussi l'opportunité de réduire la complexité des algorithmes d'inférence.

Mais ces avantages appellent des questions auxquelles nous n'avons pas de réponses définitives :

- Jusqu'où propager une décoration dans le processus de génération ? Autrement dit, à quel moment la consommer ? A priori, seule l'expérience du concepteur permet de juger de la durée de vie d'une décoration. Notre expérience nous incite à dire qu'elle ne doit plus être propagée si sa "non consommation" influe sur la génération finale. Prenons l'exemple de la figure 15 : si la suppression de tâches pour le PalmPilot nécessite la réorganisation de l'arbre des tâches, alors il n'est pas possible d'en faire une décoration. Par contre, si la suppression de tâches n'a pour effet que la disparition d'une boîte de dialogue, alors la décoration est possible et pourra être propagée jusqu'à la dernière étape de la génération ;
- Comment ré-appliquer les décorations lors d'une regénération, suite au processus itératif de développement ? Deux réponses possibles : demande explicite au concepteur pour chacune d'elles ; à défaut de spécification de la part du concepteur, les ré-appliquer ;
- Une décoration corrective peut-elle servir de décoration directive permettant ainsi d'optimiser les algorithmes de génération ? Par exemple, lors de la production des descriptions d'IHM concrète, une décoration qui force le choix d'un interacteur évite au moteur d'inférence de gérer plusieurs possibilités de représentation ;
- Lorsqu'une décoration corrective crée une divergence, est-il possible de la transformer en une décoration de factorisation afin de prolonger la partie commune ?
- Comment détecter et résoudre les conflits de décorations ? Une solution possible est d'appliquer les décorations systématiquement et de prévenir le concepteur lors d'une impossibilité :

  - soit une décoration bloque l'inférence (il n'y a pas de solution : l'énoncé est surcontraint), alors préempter le concepteur,

  - soit une décoration ne fait plus sens (elle s'applique sur un élément qui n'existe plus), alors la ranger, sans préempter le concepteur, dans un sac "Problèmes" facilement observable par le concepteur.

La problématique des incohérences et conflits est à rapprocher au cas des problèmes surcontraints, aux difficultés de prédire la surcontrainte, de montrer comment la résolution s'est faite, et quelles sont les règles à incriminer. Ce problème est décrit dans le chapitre "Génération et Inférence dans ARTStudio" à la page 129.

Avec les principes de factorisation et de décoration, nous avons traité de sujets qui relèvent de la modélisation. Avec les principes de réification et de traduction, nous abordons un second volet des outils supports : la génération.

4. Principes de Génération : Réification et Traduction

On entend par génération une succession de transformations de descriptions conduisant à des IHM multicibles exécutables. La réification et la traduction, justifiées au chapitre "De l'adaptation des IHM : analyse et taxonomies" à la page 34, constituent nos deux principes de génération. En 4.1 et 4.2, nous en affinons les définitions selon les termes de nos outils formels, puis en 4.3 et 4.4, nous analysons leurs effets sur la factorisation et la décoration.

4.1. LA RÉIFICATION

Par définition, la réification est une transformation en chose. Dans le contexte de cette étude, la réification consiste à transformer par étapes successives une description abstraite (par exemple une description des tâches) en une interface homme-machine concrète. Chaque étape produit par inférence une représentation à un niveau d'abstraction donné. Nous dirons que chaque étape définit un niveau de réification.

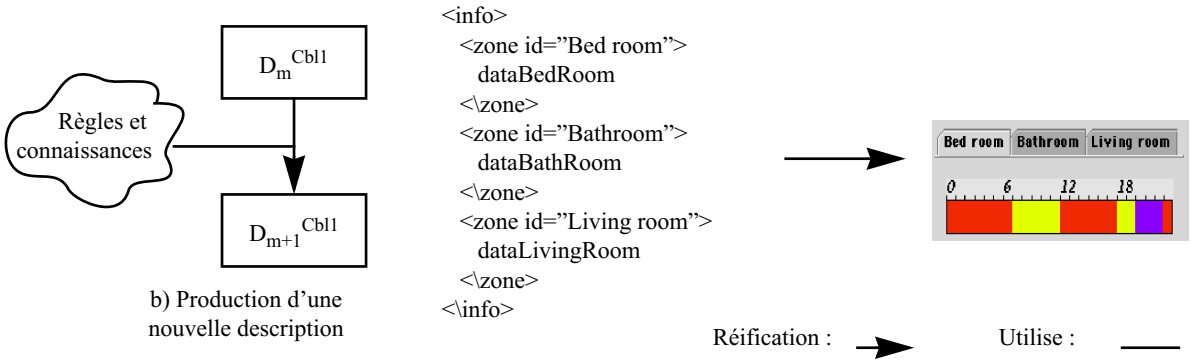
L'inférence s'exerce sur une description source : elle applique à celle-ci, un ensemble de règles de production qui exploite toutes connaissances déjà acquises par ailleurs. Les règles de production concernent l'organisation des informations, le style d'interaction, l'ergonomie, etc. ainsi que les règles de compilation de l'application.

La figure 19 illustre le principe de réification. D_m^{Cbl}, une description de *m*, modèle des concepts du domaine pour la cible *Cbl*, est réifiée en D_{m+1}^{Cbl}, une nouvelle description de *m+1*, modèle d'IHM concrète pour *Cbl*.

De manière formelle, la réification se définit comme suit.

Soit D_m^{Cbl} une description de modèle *m* pour la cible *Cbl*,
avec *Cbl = {Cbl1, ..., Cblp}*.

Figure 19. Principe de la réification. Production d'une interface graphique à partir d'une description des concepts du domaine.



$D_m^{\{Pilot,\ Mac\}}$ se décompose en :

$$D_m^{\{Pilot,\ Mac\}} \cong {}_cD_m^{\{Pilot,\ Mac\}} \tilde{\cup}\ {}_{Pilot}D_m^{\{Pilot,\ Mac\}} \tilde{\cup}\ {}_{Mac}D_m^{\{Pilot,\ Mac\}}$$

Par définition, on a : $D_m^{Pilot} \tilde{\subset} D_m^{\{Pilot,\ Mac\}}$ et $D_m^{Mac} \tilde{\subset} D_m^{\{Pilot,\ Mac\}}$.

Alors, la réification R de D_m^{Cbl} se définit ainsi :

$$R(D_m^{Cbl}) \stackrel{def}{=} \tilde{\bigcup_{\forall Cbl_i \in Cbl}} R(D_m^{Cbl_i})$$
$$\stackrel{def}{=} \tilde{\bigcup_{\forall Cbl_i \in Cbl}} D_{m+1}^{Cbl_i}$$

Par exemple, la réification R sur $D_n^{\{Pilot,\ Mac\}}$ est :

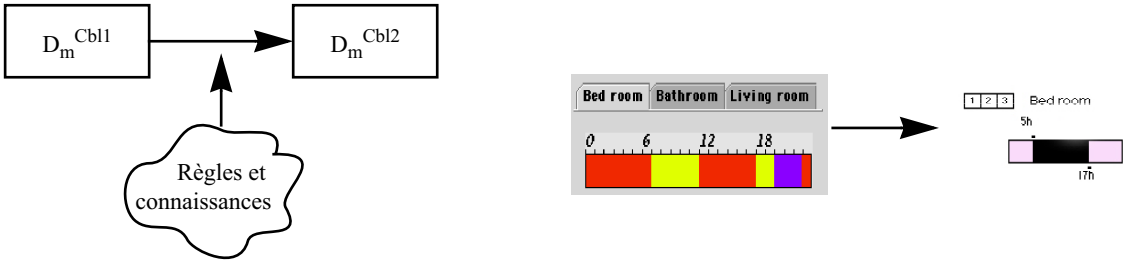
$$R(D_m^{\{Pilot,\ Mac\}}) \stackrel{def}{=} R(D_m^{Pilot}) \tilde{\cup} R(D_m^{Mac})$$
$$\stackrel{def}{=} D_{m+1}^{Pilot} \tilde{\cup} D_{m+1}^{Mac}$$

4.2. LA TRADUCTION

La traduction consiste en la reproduction d'informations pour une autre cible. À partir d'une description à un niveau de réification et pour une cible donnée, et d'un ensemble de règles, une description est produite pour une nouvelle cible. Il n'y a pas de changement de niveau d'abstraction. Par exemple (cf. figure 20) le passage d'une l'IHM concrète pour Macintosh à une IHM concrète pour PalmPilot est une opération de traduction.

De manière formelle, la traduction se note comme suit :

Figure 20. Principe de la traduction. Traduction d'une interface Mac vers une interface pour Pilot.



Soient :

- Cbl = {Cbl1, ... , Cbl n}
- $D_m^{Cbl} \cong {}_cD_m^{Cbl} \tilde{\cup} {}_{Cbl_1}D_m^{Cbl} \tilde{\cup} \ldots \tilde{\cup} {}_{Cnl_n}D_m^{Cbl}$

Alors la traduction de la description dans la cible Cbl_i ($i \in [1..n]$) vers la cible Cbl_p ($p \notin [1..n]$) se note comme suit :

$$T_{Cbl_i \to Cbl_p}(D_m^{Cbl}) \stackrel{def}{=} \left(T_{Cbl_p}(D_m^{Cbl_i}) \tilde{\cup} \tilde{\bigcup}_{\forall Cbl_j \in Cbl} D_m^{Cbl_j} \right)$$
$$\stackrel{def}{=} D_m^{Cbl_p} \tilde{\cup} \tilde{\bigcup}_{\forall Cbl_j \in Cbl} D_m^{Cbl_j}$$

4.3. GÉNÉRATION ET FACTORISATION

Nous allons étudier les effets de la réification et de la traduction au regard du principe de factorisation.

Réification et factorisation

Rappelons la définition de la réification *R* :

$$R(D_m^{Cbl}) \stackrel{def}{=} \tilde{\bigcup}_{\forall Cbl_i \in Cbl} R(D_m^{Cbl_i})$$
$$\stackrel{def}{=} \tilde{\bigcup}_{\forall Cbl_i \in Cbl} D_{m+1}^{Cbl_i}$$

Comme le montre la définition, *R* ne garantit pas la production d'une nouvelle description factorisable puisque l'intersection des D_{m+1}^{Cbli} peut-être vide.

Si l'intersection de D_m^{Cbli} est vide, nous obtenons une **divergence totale** par réification. Ce type de divergence est à mettre en opposition avec la **divergence partielle** qui correspond à une description possédant une partie commune et des parties spécifiques non vides. La divergence totale peut-être inévitable et révèle une limite du contrôle du versionnement.

Attention.

$R(D_m^{\{Pilot, PC\}}) \not\equiv R({}_cD_m^{\{Pilot, PC\}}) \tilde{\cup} R({}_{Pilot}D_m^{\{Pilot, PC\}}) \tilde{\cup} R({}_{PC}D_m^{\{Pilot, PC\}})$.

Comme le montre l'expression ci-dessus, la réification d'une description n'est pas peudo-égale à la réification de la partie commune jointe aux réifications des parties spécifiques. La réification de la partie commune est dépendante de toute la description (y compris les parties spécifiques). Il en va de même pour la réification de la partie spécifique.

Traduction et factorisation

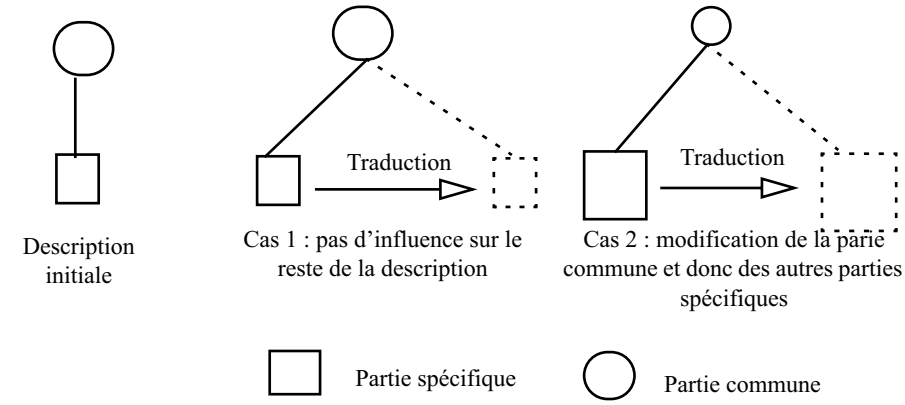
Rappelons la définition de la traduction :

$$T_{Cbl_i \to Cbl_p}(D_m^{Cbl}) \stackrel{\text{def}}{=} \left(T_{Cbl_p}(D_m^{Cbl_i}) \mathbin{\tilde{\cup}} \tilde{\bigcup}_{\forall Cbl_j \in Cbl} D_m^{Cbl_j} \right)$$
$$\stackrel{\text{def}}{=} D_m^{Cbl_p} \mathbin{\tilde{\cup}} \tilde{\bigcup}_{\forall Cbl_j \in Cbl} D_m^{Cbl_j}$$

L'application d'une traduction conduit aux deux situations suivantes :

1. Si ${}_cD_m^{\{Cbl\}} \cong {}_cD_m^{\{Cbl,\ Cblp\}}$, alors la partie commune et les parties spécifiques ne changent pas lors de la traduction (cf. figure 21, cas 1)
2. Si ${}_cD_m^{\{Cbl\}} \ncong {}_cD_m^{\{Cbl,\ Cblp\}}$, alors la partie commune diminue tandis que les parties spécifiques augmentent (cf. figure 21 cas 2).

Figure 21. Application du principe de traduction sur une description factorisée.



4.4. GÉNÉRATION ET DÉCORATIONS

Dans le cas d'une décoration corrective, la réification et la traduction ne posent pas de problème. Elles se font sur la description après interprétation de toutes les décorations correctives.

Pour les décorations de factorisation ou les corrections directives, nous nous ramenons aux points énoncées aux pages 73 et 74. Une décoration directive peut aboutir à un énoncé surcontraint et une décoration de factorisation pose le problème, que nous laissons ouvert, de sa consommation/propagation.

Dans la section qui suit nous montrons comment notre notion de description multicible, nos principes de modélisation (factorisation et décorations) et nos principes de génération (réification et traduction)

coopèrent en un cadre cohérent servant la production d'IHM multicible.

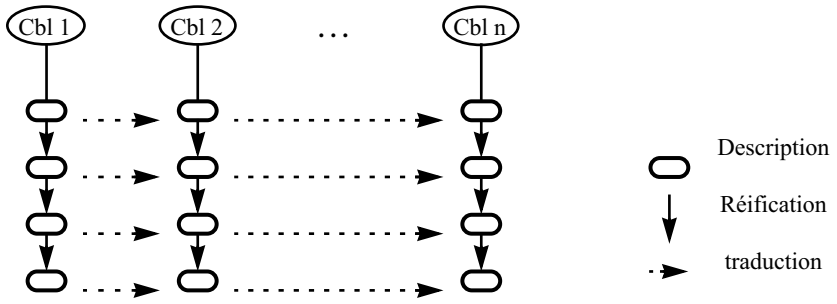
5. Cadre de référence pour la production d'IHM multicible

Le cadre que nous proposons doit tenir lieu de référence pour la définition d'outils servant la production d'IHM multicible. Nous montrons d'abord comment la réification et la traduction coopèrent (5.1) puis en 5.2, nous analysons l'apport de la factorisation. En 5.3 nous présentons une forme particulière du cadre de référence : le principe de la fermeture éclair.

5.1. COOPÉRATION ENTRE RÉIFICATION ET TRADUCTION

Comme le montre la figure 22, la génération d'IHM multicible est une combinaison de transformations de descriptions par réification et traduction. Une colonne de la figure représente, pour une cible donnée, le processus de réification d'une description. Pour une cible donnée, la réification enrichit ou produit une nouvelle description.

Figure 22. Coopération entre réification et traduction.



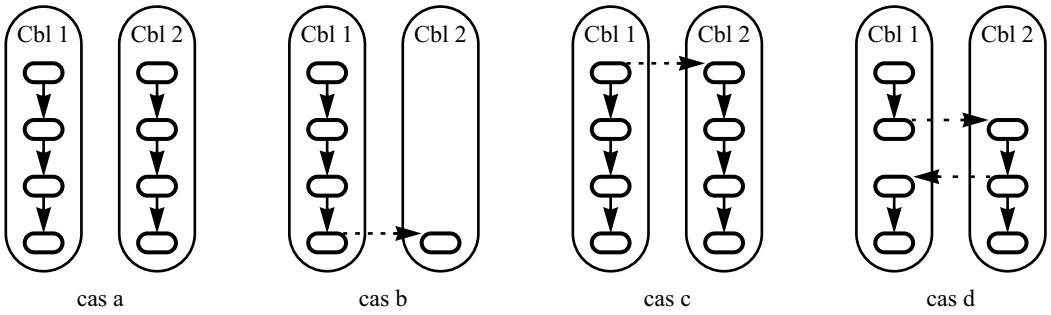
Une ligne de la figure exprime, pour une description à un niveau d'abstraction donné, la transposition de cette description d'une cible vers une autre.

Comme le montre la figure 23, la coopération entre réification et traduction peut prendre plusieurs formes :

a) Réifications indépendantes. Absence de traduction ; aucun pont entre les deux processus de génération ;

b-c) Réifications avec un seul pont de traduction. Deux cas très proches par l'unique traduction qui produit la description de démarrage à la génération pour la seconde cible. Ils se différencient par le niveau d'abstraction auquel s'effectue la traduction. Le cas b est optimum au niveau en terme de réutilisabilité puisque la spécialisation n'apparaît qu'en fin d'affaire ;

Figure 23. Différentes coopérations entre réification et traduction.



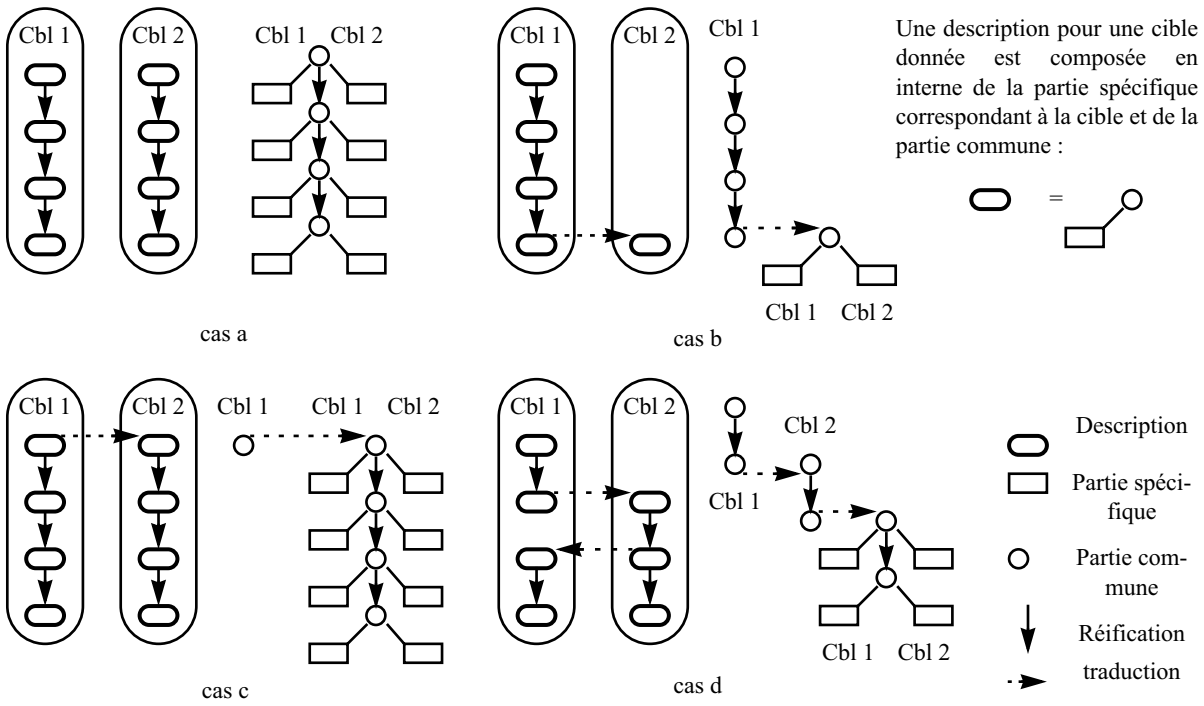
d) Réifications et traductions entrelacées. C'est un cas (théorique) complexe pour lequel des échanges ont lieu entre les deux processus de conception.

Les schémas de la figure 23 montrent l'évolution des descriptions telles que le concepteur les perçoit. Voyons les conséquences sur les représentations internes de ces descriptions en terme de factorisation.

5.2. LE PROCESSUS ET LA FACTORISATION

La figure 24 reprend les différents cas décrits dans la figure 23 :

Figure 24. Application de la factorisation. Projections des cas (a) (b) (c) et (d) de la figure 23 sur des descriptions avec factorisation.



- a). Alors que les processus de réification sont indépendants, en interne, la factorisation gère, pour chaque niveau d'abstraction, une

description commune et une partie spécifique par cible.

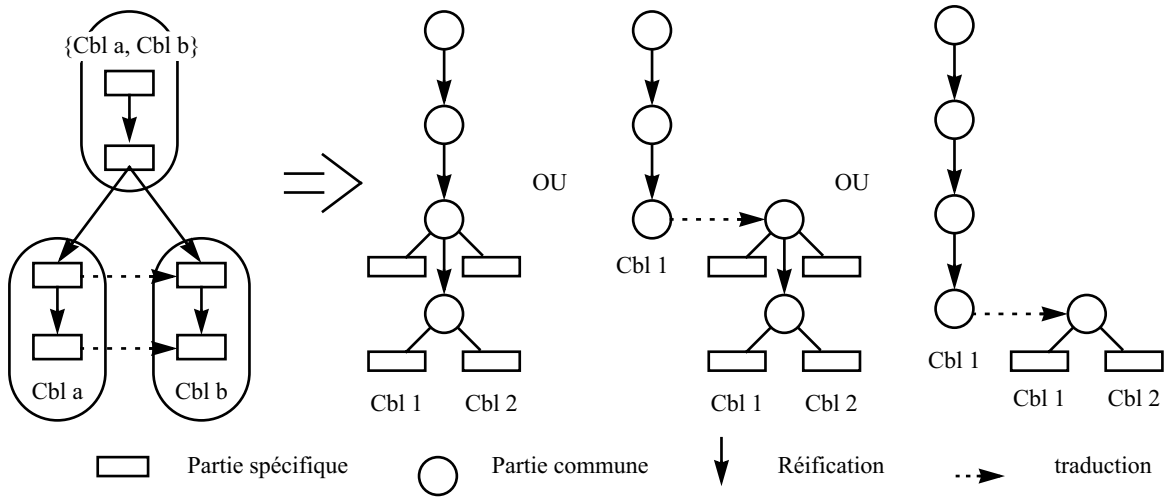
- b). Pour une traduction en fin d'affaire, la description commune est conservée jusqu'à la dernière étape. A la dernière étape, la description partage une partie commune et une partie spécifique pour chaque cible.
- c). Dans le cas d'une traduction initiale, la partie commune est transposée puis est gérée comme en a) en partie commune et parties spécifiques.
- d). Un mixage des autres cas.

Le processus de référence s'apparente à un méta-modèle de processus donnant lieu à différentes instanciations. Parmi elles, le principe de la fermeture éclair que nous retenons pour une meilleure capitalisation

5.3. LA FERMETURE ÉCLAIR

Il s'agit de retarder, au maximum, le point de divergence entre cibles. La figure 25 montre la forme avec réification et traduction, ainsi que les trois formes équivalentes avec la factorisation.

Figure 25. Processus de référence en fermeture éclair.



6. Synthèse

Dans ce chapitre, nous avons présenté un ensemble de principes motivés par la capitalisation de connaissances. Avec le principe de factorisation, nous favorisons le partage de descriptions minimisant ainsi les coûts de maintien de la cohérence. Avec le principe de décoration, le concepteur peut retarder les divergences entre descriptions, il peut regrouper les exceptions et analyser les différences entre cibles de manière centralisée. Il peut aussi déporter au niveau des outils la mémorisation des retouches de conception, inévitables dans un processus itératif.

S'il est vrai que ces principes soulèvent des problèmes théoriques non résolus, ils s'inscrivent dans un cadre de référence général dans lequel réification et traduction peuvent coopérer selon différents schémas de génération d'IHM multicible. Dans le chapitre qui suit nous montrons une application de ce cadre au cas de la production d'IHM plastiques.

Chapitre 4

Principes et processus de référence : Application à la production d'IHM plastiques

Le cadre de référence proposé au chapitre précédent fait intervenir des opérations précises (factorisation, décoration, réification et traduction), mais appliquées à des descriptions dont les éléments ne sont pas identifiés. Dans ce chapitre, nous précisons la nature des descriptions du processus de référence pour le cas particulier de la génération d'IHM plastiques[1].

Nous adoptons pour cela la démarche suivante :

- Analyse de l'état de l'art en matière de génération d'IHM,
- Identification des éléments communs entre les approches, et notamment les étapes et les descriptions impliquées,
- Instanciation de notre processus de référence général du chapitre précédent avec les étapes et description extraites de notre analyse.

La structure de ce chapitre reflète les trois étapes de notre approche.

1. Génération d'IHM : méthodes et outils

L'analyse de l'état de l'art sur les méthodes et outils pour la génération d'IHM a fait l'objet de nombreuses revues donnant lieu chaque fois à une classification particulière servant des objectifs précis [Calvary 1998][Vanderdonckt 1997]. Dans cette section nous retenons deux types d'approche qui, chacune éclaire notre quête sur la nature des modèles et étapes du processus de génération d'IHM. La première, présentée en 1.1, calque la nature des modèles sur le seul problème de la génération. La génération orientée modèle, que nous résumons en

1. Une IHM plastique, on le rappelle, est capable d'adaptation aux variations de plates-formes et d'environnements.

1.2, va plus loin en considérant tous les aspects du développement d'IHM : conception, génération, évaluation.

1.1. DÉMARCHES CENTRÉES SUR LA SEULE GÉNÉRATION

Dans cette catégorie de démarche, nous retenons les travaux de [Nanard 1990] sur la génération d'IHM révisés par [Tarby 1993] avec Diane$^+$. La figure 1 en montre une synthèse.

Principe

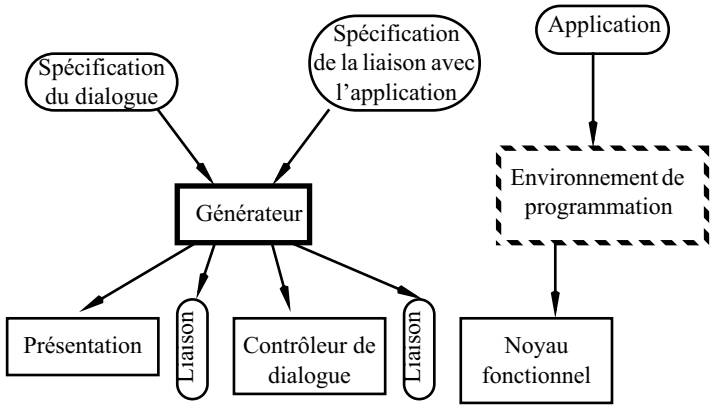
On constate que la décomposition fonctionnelle de la génération est plaquée sur les composants Arch d'un système interactif. Elle comprend la génération de la présentation, du contrôleur de dialogue, du noyau fonctionnel et des liaisons entre ces composants. Ces trois fonctions de génération sont alimentées par deux modèles spécifiant respectivement le dialogue et la liaison avec le noyau fonctionnel.

- La génération de la présentation consiste en la production de l'interface de communication entre l'utilisateur et la machine. Cette étape génère par exemple une interface graphique composée d'interacteurs, le tout étant structuré selon un modèle de présentation[1]. Les générateurs automatiques d'interfaces graphiques de type MB-IDE, proposent ce genre de processus. [Vanderdonckt 1997] fait une revue des différentes approches ;
- La génération du contrôleur de dialogue consiste en la construction de l'enchaînement du dialogue, la gestion des événements externes (venant ou en direction de l'utilisateur) et internes (venant ou en direction du système). Ce dialogue peut-être explicite (externe pour Tarby) ou implicite (interne pour Tarby). Dans le premier cas, un module dédié au dialogue est généré alors que dans le second cas, le dialogue est mélangé avec le reste de l'application et en particulier avec le code de la présentation. Il est préférable pour des raisons de portabilité, de réutilisabité ou de clarté (pour le développeur) d'adopter un dialogue explicite. Le code du contrôleur de dialogue peut-être produit à partir de plusieurs types de spécification à l'aide par exemple, de langages graphiques, de réseaux de Petri ou de langages textuels.
- La génération du noyau fonctionnel consiste à produire le code du noyau fonctionnel. Nous ne nous intéresserons pas à cet aspect dans notre thèse.
- La génération des liaisons. Une liaison est un adaptateur entre composants, voire un connecteur. Très souvent, ces modules de liaison sont intégrés au contrôleur de dialogue et ne figurent pas dans un

1. Nous reviendrons sur le modèle de présentation. Succinctement, il s'agit d'un ensemble de règles de placement spatio-temporel entre objets d'interaction.

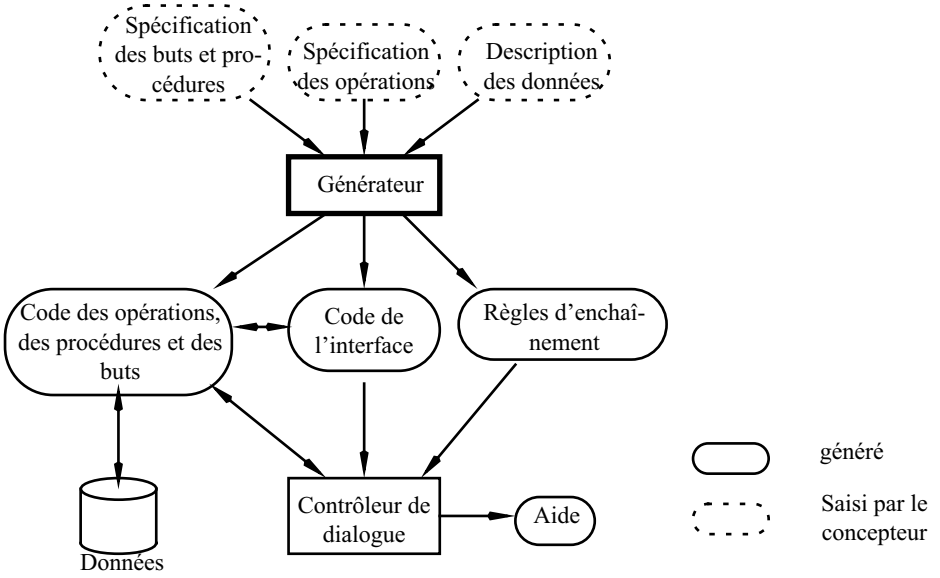
module explicite.

Figure 1. Démarche pour la génération d'interface. Issue de [Nanard 1990] et [Tarby 1993].



Diane^{+} Diane^{+} [Tarby 1993] repose sur une extension de la méthode de conception Diane. La figure 2 montre la démarche dans son ensemble. Elle est semblable, par l'esprit, à l'approche de la figure 1 mais diffère par la nature du contrôleur de dialogue. Le composant "Code de l'interface" de la figure 2 correspond à la "Présentation" de la figure 1. Chez Tarby, le contrôleur de dialogue est en fait un moteur d'inférence qui interprète les règles d'enchaînement pour produire la dynamique du dialogue, c'est-à-dire l'appel dynamique du code des buts, procédures et opérations (ces dernières correspondant à des tâches élémentaires). Ces règles d'enchaînement, buts, procédures et opérations sont spécifiés selon les préceptes de la méthode Diane.

Figure 2. Démarche de conception dans Diane Tarby [Tarby 1993].



Les travaux que nous venons de présenter, déjà anciens, se préoccupaient du seul problème de la génération de code. Mais la génération

n'est qu'une étape du processus de développement d'IHM. Les générateurs orientés modèles présentés ci-dessous vont plus loin.

1.2. GÉNÉRATEURS ORIENTÉS MODÈLE (GOM)

Les Générateurs Orientés Modèle (GOM)[1] visent la couverture de tous les aspects du développement d'IHM : conception, génération et évaluation. Dans ce but, ils intègrent de multiples connaissances au sein de descriptions qui alimentent un ensemble cohérent d'outils de conception, d'évaluation et d'implémentation [Sukaviriya et al. 1994].

Les descriptions d'un GOM sont les suivantes :

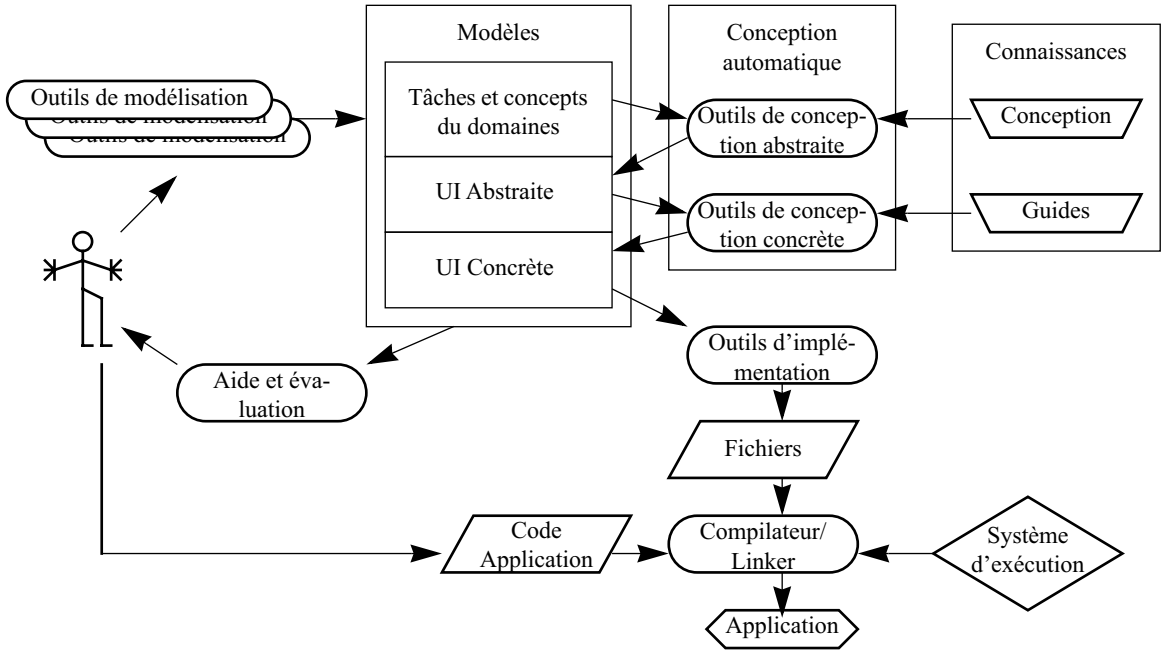
- Modèle de l'utilisateur : recouvre les caractéristiques des utilisateurs visés (âge, profession, expérience, etc.). C'est un modèle important puisqu'il permet de créer des interfaces adaptées ou adaptables aux futurs utilisateurs ;
- Modèle de l'environnement : définit le contexte d'utilisation du logiciel (par exemple, au bureau, chez soi ou sur le terrain) ;
- Modèle des tâches : ce terme est utilisé différemment selon les cultures scientifiques. Ici, il s'agit de la spécification des tâches que l'utilisateur désire accomplir. Ce modèle se déduit de l'analyse de l'activité ;
- Modèle du domaine : spécifie les fonctions et les objets réalisés dans (ou exportés par) le noyau fonctionnel ;
- Modèle du dialogue : définit la structure du dialogue entre l'utilisateur et la machine. Il est directement dérivé du modèle de tâches et du modèle du domaine. Ce modèle définit la structure opératoire de réalisation des tâches avec le système informatique ;
- Modèle de l'application : décrit la sémantique du noyau fonctionnel et les services qu'il assure ;
- Modèle d'ergonomie : définit les règles d'ergonomie pour la construction d'interface. Ces règles peuvent être utilisées au cours de la génération ou lors d'une évaluation après la génération ;
- Modèle de présentation : représente le composant d'Interaction du modèle Arch. Il peut être écrit avec un langage de haut niveau ou en Tk par exemple ;
- Modèle de comportement : décrit le comportement des entrées. Il se situe à différents niveaux. Il peut décrire de manière très formelle le comportement des objets d'interaction ou spécifier le positionnement relatif des fenêtres ;
- Modèle de la plate-forme : définit les caractéristiques du système ou des systèmes visés (spécification des dispositifs d'entrées et de sorties). C'est un modèle très important pour notre problème, mais, à

1. GOM : notre traduction de *Model-Based Interface development environment* (MB-IDE)

notre connaissance, il n'a pas été utilisé.

Cette liste de descriptions se veut exhaustive et idéale. En pratique, aucun GOM ne les inclut toutes. Chaque GOM pioche dans cette liste, celles qui correspondent aux objectifs visés. [Szekely 1996] en propose une instanciation pratique illustrée par la figure 3. On y trouve :

Figure 3. Structure pour un MB-IDE. Tiré de [Szekely 1996] pp.3.



Le composant "Modèles". C'est le composant principal du système. Il représente l'IHM selon trois niveaux d'abstraction, niveaux que les démarches de la section 1.1 n'avaient pas clairement identifiés.

- Le plus haut niveau regroupe les concepts du domaine et le modèle de tâche véhiculé par le système.
- Le niveau intermédiaire, l'"*abstract user interface spécification*", sert à spécifier : a) les tâches élémentaires d'interaction, par exemple la navigation dans une liste déroulante (popup list), et b) les concepts du domaine à rendre observables.
- Le niveau "*concrete user interface specification*", spécifie le style d'affichage, de rendu des unités du niveau intermédiaire. Il correspond au niveau présentation de Nanard.

Utilitaires de spécification du modèle. Les utilitaires de spécification assistent le développeur dans la spécification des modèles. Leur intérêt principal est de cacher la partie langage de spécification en procurant une interface graphique pour la modélisation. Szekely pense que le peu de succès des GOMs dans l'industrie vient de la complexité de

ces langages. Ces utilitaires de spécification ont donc une grande importance pour la simplification de l'utilisation d'un GOM.

Utilitaires d'aide et d'évaluation. Les utilitaires d'aide et d'évaluation permettent d'évaluer la conception. L'approche GOM est très appropriée à ce genre d'utilitaires puisqu'elle rassemble une grande quantité de connaissances sur l'utilisateur, le noyau fonctionnel, le contexte d'exécution, etc. Les utilitaires d'évaluation contiennent une base de connaissances sur les règles de conception.

Utilitaires de construction automatique. Les utilitaires de construction automatique permettent, à partir des spécifications de modèles et de règles heuristiques ou "*guidelines*", de construire automatiquement l'interface graphique. Selon les produits, nous observons différents niveaux d'interaction avec le développeur. Dans Mastermind, par exemple, une partie de la présentation est faite par le développeur alors que dans TRIDENT la génération est complètement automatisée.

Logiciel d'implémentation. Le logiciel d'implémentation traduit les spécifications concrètes de l'interface en langage directement exécutable ou utilisable par un constructeur d'interface. Par exemple Mastermind génère du code source C++. D'autres, tel que FUSE, génèrent un script utilisable par un autre constructeur d'interface. ITS ou Humanoid, quant à eux, interprètent directement les spécifications par un système de "*run-time*". Certains GOMs, tel UIDE (rappelé dans [Dieterich et al. 1993]) génèrent automatiquement une partie de l'aide.

1.3. SYNTHÈSE

Les démarches présentées dans cette section ont en commun la description du modèle des tâches et des concepts du domaine. Elles diffèrent par leur structure et leur couverture du processus.

Diane$^+$ s'appuie sur le modèle Arch [Bass 1991] pour recommander la génération de quatre composants (Noyau Fonctionnel, Contrôleur de dialogue, Présentation et Adaptateurs). Partant de là, Diane$^+$ introduit les descriptions nécessaires à la génération de ces quatre composants et seulement cela.

Les GOMs adoptent une démarche de construction d'interface en distinguant des étapes auxquelles correspondent des niveaux d'abstraction spécifiques (le niveau tâches&concepts, IU Abstraite, IU Concrète, Fichiers de code, et exécutable).

2. Identification des étapes et des descriptions

Nous le disions, un processus de génération d'IHM comprend des étapes. Chaque étape produit des résultats à partir des résultats d'étapes antérieures. Ces résultats sont concrétisés sous forme de descriptions. Ces deux points, étapes et descriptions, sont détaillés maintenant à la lumière de l'apport de l'état de l'art de la section 1.

2.1. ETAPES DE GÉNÉRATION

La génération d'IHM comprend trois étapes correspondant chacune à un niveau d'abstraction de description :

1. La première étape, celle du niveau le plus haut, correspond à la définition des tâches avec leur décomposition et relations temporelles et cela en relation avec les concepts du domaine manipulés au cours de ces tâches.
2. La deuxième étape est l'extraction des espaces de travail à partir des modèles des tâches et concepts (résultats de l'étape précédente). Un espace de travail est une unité abstraite de présentation qui regroupe en son sein tout ce qui est nécessaire à la manipulation d'un ensemble logique de concepts, ou qui est nécessaire à l'accomplissement d'un ensemble logiquement connecté de tâches [Normand 1992]. De multiples facteurs — de conception, d'ergonomie, et d'heuristiques pas toujours bien formalisées— prévalent à l'identification des espaces de travail. Notre concept d'espace de travail correspond à celui de Fenêtre Logique et d'Unité de Présentation de TRIDENT [Vanderdonckt 1997]. Ce concept n'existe pas explicitement dans Diane$^+$ mais est à rapprocher de la notion de But principal.
3. La troisième étape, du niveau d'abstraction le plus bas, consiste à produire l'IHM concrète. Ici, sont choisies les techniques de navigation qui vont permettre à l'utilisateur final de migrer entre espaces de travail, de même le choix des interacteurs qui représenteront les concepts du domaine et permettront de réaliser les tâches élémentaires.

Notons que ces trois étapes de génération, qui définissent en séquence trois niveaux d'abstraction, sont par définition reliées par le principe de réification. Disposant des étapes, nous sommes en mesure d'identifier les descriptions correspondantes.

2.2. LES DESCRIPTIONS

Nous distinguons trois types de description : les descriptions initiales, transitoires et terminales.

Descriptions initiales

Une description est initiale si elle remplit les deux conditions suivantes :

- elle est produite par le concepteur
- elle sert seulement d'entrée à une étape.

Pour le développement d'IHM plastique, nous proposons les descriptions initiales suivantes :

- Le modèle des concepts,
- Le modèle des tâches,
- Le modèle de la plate-forme — caractéristiques physiques et logiques (interacteurs...),
- Le modèle de l'environnement,
- Les règles d'ergonomie et les heuristiques métiers.

Les descriptions initiales constituent le dépôt de connaissances. Elles sont établies manuellement, capitalisées puis référencées dans le processus de développement lors d'une opération de réification ou de traduction. Elles sont susceptibles d'intervenir dans toutes les étapes de génération comme connaissances d'entrée.

Descriptions transitoires

Une description est transitoire si elle est le fruit d'une étape intermédiaire de réification ou de traduction. Une description transitoire peut être produite par le concepteur, ou générée par l'outil, voire produite par l'outil et retouchée par le concepteur.

Concernant la génération d'IHM plastique, nous proposons les trois descriptions intermédiaires suivantes :

- La description "tâches&concepts" qui résulte d'un enrichissement du modèle des tâches. Elle précise, pour chaque tâche, les concepts manipulés. Elle décore les tâches d'attributs de criticité, de complexité, de fréquence, etc., pour le contexte d'interaction traité. Pour ce faire, elle référence les descriptions initiales modèle de tâches et modèle des concepts. Cette description fournit les premières briques au processus de génération ;
- L'interface abstraite qui structure l'IHM en espaces de travail et spécifie l'enchaînement entre espaces ;
- L'interface concrète qui concrétise les espaces de travail en termes d'interacteurs.

Descriptions terminales

Une description est terminale si elle est le fruit final du processus de génération.

Les descriptions terminales constituent tout ou partie du système exécutable. Dans Diane$^+$, les composants "code des opérations", "code de l'interface", "règles d'enchaînement" sont des descriptions terminales.

Disposant des étapes et des descriptions, nous pouvons instancier notre cadre de référence pour le cas des IHM plastiques.

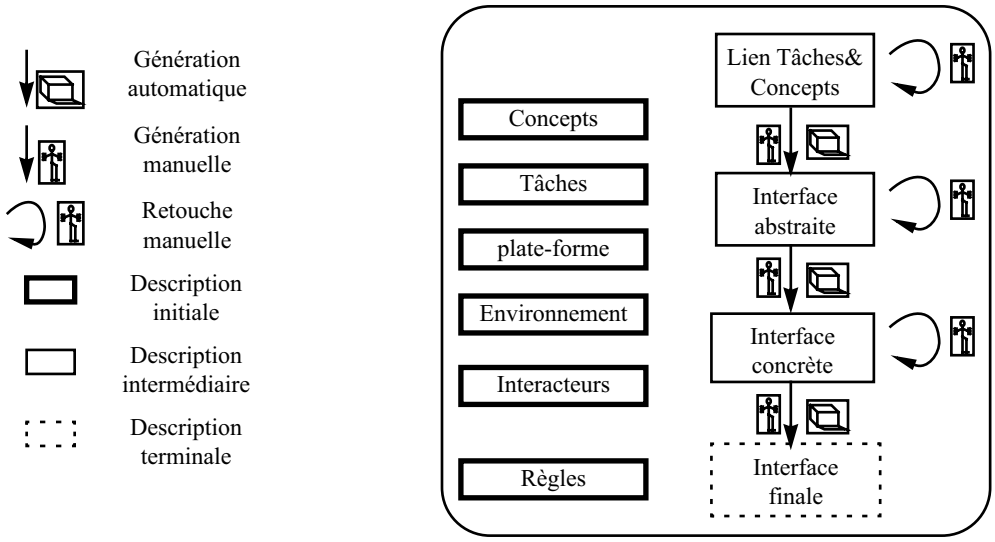
3. Instanciation du cadre de référence

Nous allons présenter l'instanciation du cadre de référence en trois étapes : réification seulement, puis coopération réification et traduction, enfin la fermeture éclair.

3.1. RÉIFICATION SEULEMENT

La figure 4 est une instanciation de notre cadre de référence où seule est présentée la réification. On obtient une représentation d'un générateur GOM de base.

Figure 4. Cadre de référence pour la génération d'IHM plastique. Cas de la réification.



Nous proposons un processus de réification en quatre étapes impliquant les descriptions suivantes comme suit :

- Les descriptions initiales (concepts, tâches, plates-formes, environnements, interacteurs et règles) sont susceptibles de servir les trois étapes de réification ;
- Le résultat de la première étape est la description transitoire "Tâches&Concepts" ;
- "Tâches&Concepts", aidée de tout ou partie des descriptions initiales, sert d'entrée à l'étape de réification suivante qui produit la des-

cription transitoire "interface abstraite" ;

- "Interface abstraite", avec tout ou partie des descriptions initiales, est réifiée pour produire la description transitoire "interface concrète" ;
- "Interface concrète" est à son tour réifiée en la description terminale "interface finale" ou code exécutable.

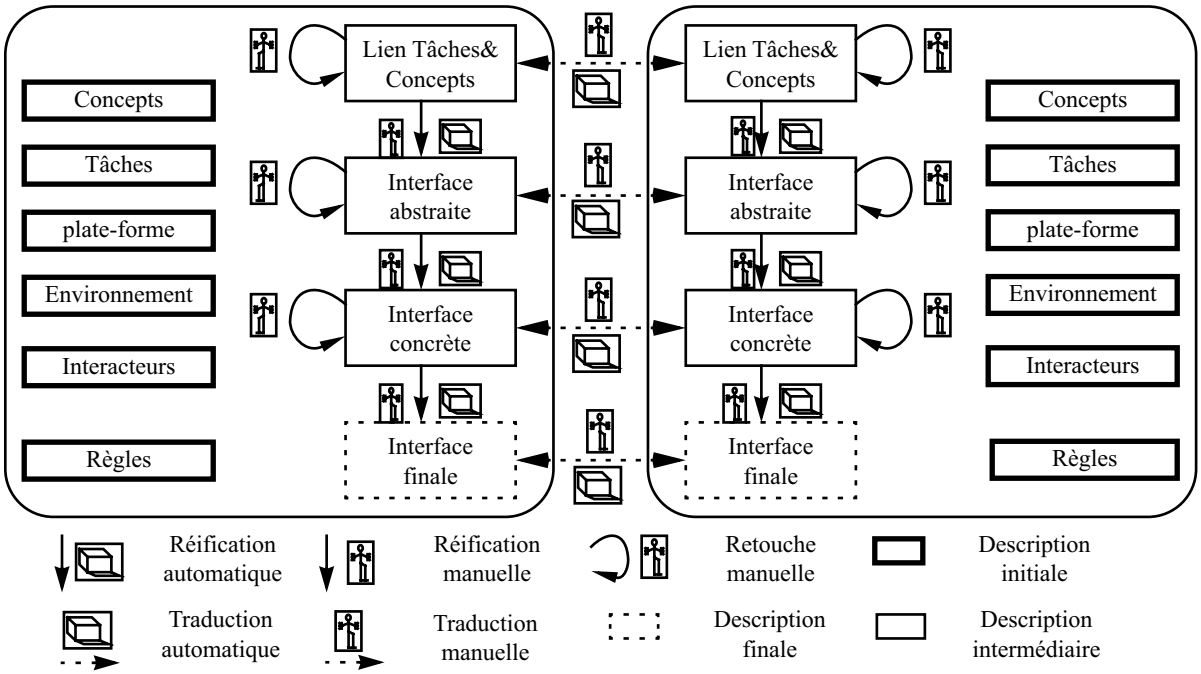
Attention. Il est important de noter que chaque étape de réification est assurée :

- soit par l'outil ;
- soit par le concepteur (en l'absence d'outil adapté) ;
- soit par la coopération de l'outil et du concepteur (l'outil produit et le concepteur ajuste). Cette dernière option est à notre avis optimale puisque l'outil prend en charge les cas standard laissant au concepteur le soin de produire les expressions subtiles trop lourdes à formaliser dans un canevas général.

3.2. Coopération Réification et Traduction

La figure 5 présente le cadre dans son ensemble où réification et traduction coopèrent. On relèvera que là encore, une traduction peut être assurée par l'outil, ou bien effectuée par le concepteur ou bien résulter de la coopération des deux acteurs.

Figure 5. Instanciation du cadre de référence pour la génération d'IHM plastique.

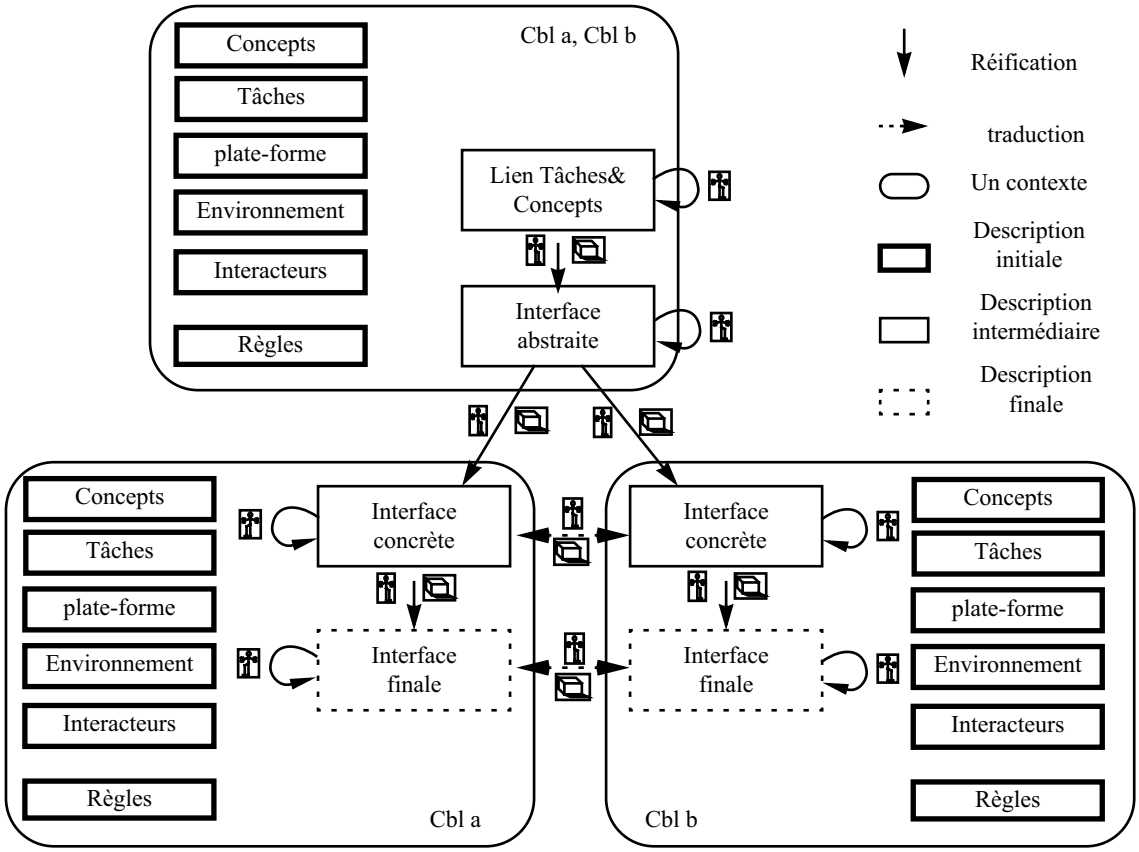


3.3. La fermeture éclair

La figure 6 montre le processus de génération en fermeture éclair déduit de l'instance générale de la figure 5. C'est ce processus que nous avons appliqué pour ARTStudio.

On constate que les deux premières étapes de réification sont multicibles. Une divergence apparaît au niveau de la production de l'interface concrète : l'interface abstraite, comme son nom l'indique, a de bonnes raisons d'être indépendante de la cible. Une interface concrète, comme son nom l'indique, est proche de la cible. Il est donc raisonnable que la réification de l'interface abstraite donne naissance à deux interfaces concrètes à raison d'une description par cible.

Figure 6. Processus en fermeture éclair pour la génération d'IHM plastique.



Nous achevons ici notre présentation de la seconde partie du mémoire. Au chapitre 3, nous avons posé les principes et un cadre de référence pour la génération d'IHM multicible, puis, dans ce chapitre, nous avons donné corps à ce cadre en précisant la nature des descriptions mises en jeu de même que les niveaux de réification. La dernière partie du mémoire montre comment nous avons mis en pratique nos propositions. Nous commençons par détailler, au chapitre suivant, les éléments des descriptions implémentées dans ARTStudio.

Partie 3 Implémentation

Chapitre 5 *Les descriptions dans ARTStudio*

ARTStudio est un assistant logiciel pour le développement d'IHM plastique. Il est constitué d'outils (éditeurs, gestionnaires, générateurs) coopérant selon l'exemplaire de fermeture éclair présenté au chapitre précédent. A cette occasion, nous avons introduit les descriptions impliquées dans la fermeture éclair sans toutefois en préciser le contenu.

Ce chapitre a pour objet de concrétiser les descriptions de ARTStudio. Pour chacune d'elles, nous précisons ses éléments et le formalisme d'expression en justifiant notre choix. Nous passons successivement en revue :

- Parmi les descriptions initiales : le modèle des concepts du domaine, le modèle de tâche et le modèle des interacteurs.
- Parmi les descriptions transitoires : le modèle des Tâches&Concepts l'Interface Utilisateur Abstraite (IUA) et l'Interface Utilisateur Concrète (IUC).
- Pour les descriptions finales : L'Interface Utilisateur Finale (IUF)

On trouvera en annexe 2 le détail des structures internes utilisées par les outils de ARTSTudio, de même que l'interface homme-machine de ces outils. Les mécanismes d'inférence des descriptions transitoires et finales ne sont pas traitées ici : elles font l'objet du chapitre suivant.

1. Modèle des concepts du domaine

Le modèle des concepts du domaine décrit l'ensemble des concepts manipulés par le noyau fonctionnel en vue de les présenter à l'utilisateur ou de permettre un lien sur les fonctions du noyau fonctionnel. Dans ce paragraphe, nous allons tout d'abord définir nos requis sur la modélisation des concepts puis couvrir un aperçu de l'existant dont l'analyse conduira notre choix de formalisme.

1.1. REQUIS

Un concept doit être manipulable (par l'utilisateur). Par conséquent, notre premier requis s'exprime comme suit :

1. le formalisme d'expression doit décrire les fonctions applicables aux concepts.

Un concept doit être représentable (auprès de l'utilisateur). Or, il peut arriver qu'aucun interacteur ne soit capable de représenter un concept donné. Par exemple, les bibliothèques graphiques usuelles n'offrent pas l'interacteur qui permettrait de représenter le concept de date. Or ce concept peut être représenté par une composition d'interacteurs qui, eux, sont disponibles. Cet exemple nous amène à définir trois requis supplémentaires sur la structure des concepts et leur domaine de valeur :

2. Le formalisme d'expression doit permettre de décrire les relations de composition d'un concept. Un concept composé est dit **complexe**. Dès lors, il est possible de le représenter par le biais de ses attributs qui à leur tour, peuvent être décomposés ou représentés par des interacteurs.
3. Le formalisme d'expression doit permettre la description des relations d'héritage. Ainsi un interacteur représentant un concept générique peut également représenter un concept spécifique.
4. Le formalisme d'expression doit permettre de décrire le domaine de valeur d'un concept. Alors il est possible de choisir un interacteur proactif ou assurer une réactivité[1] de meilleure qualité par le biais de la délégation sémantique.

1.2. LANGAGES D'EXPRESSION

Comme le montre la table 1, nos quatre requis permettent de résumer les capacités d'expression de différents langages de modélisation de concepts. Nous avons retenu : IDL, les travaux de Quesnot, UML, XML et XML-Schema, XTL, Adaptive Form, les langages de représentation de connaissances, et SAGE.

Table1. Caractérisation des langages d'expression pour la modélisation de concept.

● répond au requis ○ ne répond pas au requis ◐ répond partiellement au requis

	Fonction	**Relation de composition**	**Relation d'héritage**	**Domaine de valeur**
Nom	●	●	◐	○

1. En Interaction Homme-Machine, la proactivité vise à empêcher l'erreur humaine (les éléments de menu en grisé clair en sont un exemple), tandis que la réactivité se veut corrective (l'erreur humaine ne peut être empêchée, et si elle est commise, elle est signalée avec les messages d'explication et de recommandation. Se référer à la thèse de Gaëlle Calvary [Calvary 1998] pour plus de détail sur le sujet.

IDL IDL (*Interface Description Language*) s'inscrit dans la mouvance CORBA [OMG 1991]. Il permet de définir l'interface entre un client qui utilise un service et l'implémentation de ce service. Sa syntaxe est inspirée de celle de C^{++}. La figure 1 montre un sous-ensemble de la grammaire qui montre la possibilité de décrire des concepts complexes sans toutefois permettre de représenter leur domaine de valeur. IDL, qui n'est pas orienté objet, ne permet pas la description de la relation d'héritage.

Figure 1. Sous-ensemble de la grammaire d'IDL. IDL permet de décrire des concepts complexes mais sans spécifier leur domaine de valeur.

<type_dcl>	::=	"typedef" <type_declarator> | <struct_type> | <union_type> | <enum_type>
<struct_type>	::=	"struct" <identifier> "{" <member_list> "}"
<member_list>	::=	<member>+
<member>	::=	<type_spec> <declarators> ";"
<type_spec>	::=	<simple_type_spec> | <constr_type_spec>
<constr_type_spec>	::=	<struct_type> | <union_type> | <enum_type>

Quesnot [Quesnot 1995] Quesnot propose un langage de spécification de l'interface fonctionnelle d'une application qui permet de décrire la structure des données et les opérations qui leur sont applicables. Ainsi un objet (CONCEPTUAL TYPE figure 2) est décrit comme étant composé d'un ou plusieurs attributs, et d'opérations. Ce langage ne répond pas parfaitement à nos requis puisqu'il ne permet pas de décrire explicitement des concepts complexes. En effet, une classe "CONCEPTUAL TYPE" possède des attributs de type "TYPE". Un attribut "TYPE" peut-être décrit par un constructeur : SET, SEQ, UNION, TBL, ENUM. Mais les éléments ne sont pas dénommés et sont en général de même types et simples (booléen, entier, etc.). Aussi n'est-il pas possible de décrire une donnée du type "une voiture est composée de quatre roues et d'un moteur" et "une roue est composée de quatre ou cinq boulons, d'un pneu et d'une jante".

UML [Muller 1999] UML est une méthode d'analyse et de conception orientées objets. La notation UML permet de décrire un ensemble d'informations sous la forme de diagrammes dont les diagrammes de classes. Les diagrammes de classes expriment la structure statique des concepts du domaine en termes d'attributs, de méthodes et de relations entre objets. Ces relations sont de types association/agrégation et relation d'héritage. La figure 3 montre un exemple de diagramme UML. La notation offerte

Figure 2. Syntaxe de l'interface fonctionnelle d'une application. Extrait de Quesnot [Quesnot 1995].(1) Indique que la grammaire continue, mais que nous n'allons pas plus loin dans les détails.

<CONCEPTUAL TYPE> ::= **"TYPE"** <TYPE IDENT>
[**"DESCRIPTION"** <DESCRIPTION>]
"Attributes" <ATTRIBUTE DECL> { **";"** <ATTRIBUTE DECL> }*
[**"Operation"** <OP DECL> { **";"** <OP DECL> }*]
"END TYPE"

<TYPE> ::= <TYPE INDENT> |
"SET" "[" <TYPE> **"]"** |
"SEQ" "[" <TYPE> **"]"** |
"UNION" "[" <TYPE> **","** <TYPE> **"]"** |
"TBL" "[" <TYPE>**","** <TYPE> **"]"** |
"ENUM" "[" <TERM> { **","** <TERM> }* **"]"** |
<BASIC TYPE>

<BASIC TYPE> ::= **"BOOLEAN"** |
"INTEGER" |
"REAL" |
"CHAR" |
"STRING" |
"TEXT" |
"NIL" |

<TYPE INDENT> ::= <INDENT>

<ATTRIBUTE DECL> ::= <NAME> : <TYPE>

<NAME> ::= [a-z] [a-z0-9_-]*

<INDENT> ::= [A-Z] [A-Z0-9_-]*

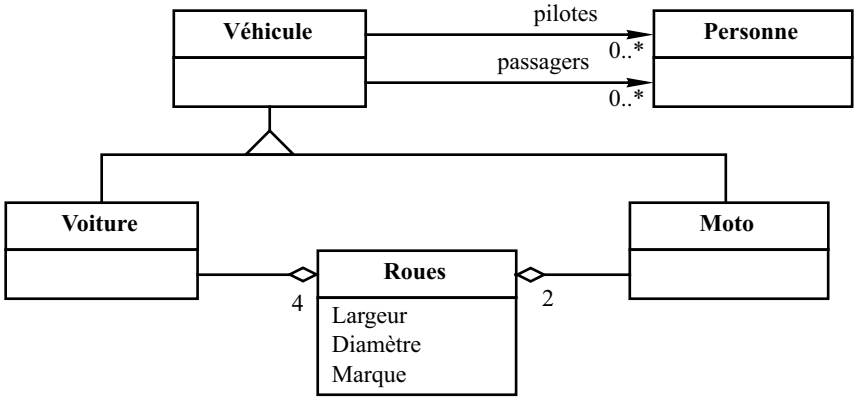
<OP DECL> ::= **... (1)**
"op decl" décrit l'interface des procédures (nom, paramètres d'entrées et de sorties)

<DESCRIPTION> ::= **???**
"description" est non décrit dans [Quesnot 1995]. Cela semble être une description textuelle à but informatif.

<TERM> ::= **... (1)**
"term" est une valeur terminale. par exemple : 1, 2, jeudi, jean.

par les diagrammes de classes répond aux requis 1, 2 et 3 mais pas à la description des domaines de valeur. Pour combler ce manque, IBM a contribué à l'extension d'UML en proposant le langage de contrainte OCL (*Object Constraint Language*) [IBM OCL]. Il est intégré dans la norme 1.1 d'UML. Ainsi UML 1.1 répond à nos requis.

Figure 3. Exemple de diagramme de classes UML. Une moto et une voiture sont des véhicules, ayant respectivement 2 et 4 roues. Un véhicule transporte éventuellement des passagers et a normalement au moins un pilote (sauf le Météor!)



XML et XML-SCHEMA XML [Bray et al. 2000] est un métalangage conçu dans le but de décrire des données structurées. C'est un sous-ensemble de SGML, utilisant un système de balise pour décrire une structure arborescente

d'un document. Fondé sur XML, XML-SCHEMA permet d'étendre le typage des éléments proposés par XML, dans une DTD standard. Il introduit de nouveaux types numériques, dates, types énumérés, des intervalles et d'autres types arbitraires [Biron 2001], alors que XML ne propose que le type #PCDATA. Il autorise la définition de nouvelles structures [Thomson et al. 2001] pour les documents et propose des mécanismes d'expression d'héritage entre types d'éléments et/ou types nouvellement introduits.

La figure 4 montre une manière de décrire un concept complexe, composé d'attributs simples ou complexes. Ici le type Publication est composé de trois attributs : Titre, Auteurs et Date de la publication.

Figure 4. Exemple de description avec XML Shema.

```
<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Titre" type="xsd:string" minOccurs=1 maxOccurs=1/>
    <xsd:element name="Auteurs" type="xsd:string" minOccurs=1 maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:year"/>
  </xsd:sequence>
</xsd:complexType>
```

A la différence des langages comme Java ou C++ pour lesquels il existe un seul type d'héritage (il peut être simple ou multiple) XML Schema propose différentes sémantiques. Il propose l'héritage :

- par restriction du domaine de valeur de la super-classe : une note est un entier appartenant à l'intervalle [0, 20]. Cette restriction peut s'appliquer à un concept complexe ;
- par extension d'une superclasse : une moto est un véhicule utilisant deux roues. Correspond à l'héritage simple du Java ou C++ ;
- par union d'un ensemble de superclasses : un Mois est un Entier et une Chaîne de caractères. Correspond à l'héritage multiple dans C^{++} ;
- par liste d'un ensemble de superclasses : un livre est une liste de chapitres. Chapitre est une superclasse de Livre.

Les schémas XML proposent une solution normalisée et complète pour la description de données (structure et domaine de valeur). Par contre, de par leur objectif de spécification de documents, ils ne permettent pas la description des fonctions.

XTL [Brun 1998] XTL pour (*eXtended Temporal Logic*) est un langage pour la spécification formelle des systèmes interactifs. Le langage proposé reste très simple sur la description des types de concepts manipulés : ce n'est pas l'objectif visé. Les variables sont définies par un type de données et un domaine de validité. Il utilise six types de données : le booléen, l'entier, le réel, la chaîne de caractères, la liste et l'ensemble. Les struc-

Figure 5. Exemple de spécialisation dans XML Shema. Spécialisation par restriction, extension ou union.

```
<xsd:simpleType name= "NoteDExamen">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="20"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="Livre">
  <xsd:complexContent>
    <xsd:extension base="Publication" >
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Editeur" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:simpleType name=”Mois”>
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="1"/>
        <xsd:maxInclusive value="12"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="janvier"/>
        ...
        <xsd:enumeration value="décembre"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

tures complexes de données ne sont pas décrites, pas plus que l'héritage.

Adaptive form *Adaptative form* [Frank 1998], est un outil de génération d'interfaces graphiques à partir d'une grammaire. Cette grammaire décrit un arbre représentant des concepts structurés. Les interfaces générées sont de type formulaire. Avec *Adaptative form* il n'est pas possible de décrire des fonctions du noyau fonctionnel et les relations d'héritage.

Langages de représentation de connaissances Dans le domaine de la représentation de connaissances, un certain nombre de travaux ont été effectués dans le but de représenter formellement la syntaxe mais aussi la sémantique de données. La logique terminologique, la logique assertionnelle, les graphes conceptuels [Euzenat 1998] [Ducournau 1996] sont autant de langages permettant de décrire des structures, du typage, des domaines concrets (entiers, réels, etc.) et des domaines abstraits (les objets, termes, etc.) et d'attribuer une sémantique aux expressions. Aucun d'eux ne couvre tous nos requis, en particulier la description des fonctions du noyau fonctionnel.

SAGE et Autres La caractérisation des données dans le système SAGE [Roth 1990] est construite de manière à permettre la génération de présentations graphiques d'information de type “business” ou statistique. Elle met en avant plusieurs critères permettant de faire une association entre les données et différents types de représentations graphiques, permettant l'ordonnancement de l'information, etc. Il existe d'autres types de description de données comme celle de Bruley [Bruley 1999] qui caractérise les données selon des dimensions — 1D, 2D, ..., nD, selon une structure particulière — hiérarchique, temporelle, etc. Ces outils ne répondent pas à nos requis puisqu'ils sont empreints d'une sémantique pour la représentation de données selon un point de vue graphique par-

ticulier — la visualisation de grandes quantités d'information — et n'ont pas comme objectif la description de structures ou des fonctions applicables à ces données.

1.3. SYNTHÈSE

La table 2 présente un résumé des différents langages de description de concepts. Par rapport à nos requis, UML 1.1 répond à nos attentes. Les schémas XML, bien qu'ils n'autorisent pas l'expression des fonctions du NF, couvrent également très bien nos requis. Ces deux notations sont très intéressantes également par leur ouverture et leur normalisation. Les langages de représentation de connaissances offrent une puissance d'expressivité (la sémantique en particulier) supérieure à UML ou XML-Schema. Mais compte tenu de leur complexité, nous laisserons de côté ce type d'approche.

Table2. Caractérisation des formalismes de description de concepts

	Fonction	Relation de composition	Relation d'héritage	Domaine de valeur
IDL	●	●	○	○
Quesnot	●	◐	◐	●
UML	●	●	●	● a
XML Schema	○	●	●	●
XTL	●	○	○	●
Adaptive form	○	●	○	●
Représentation de connaissances	○	●	●	●
Sage et autres	○	◐	○	●

a. Possible à partir de la version 1.1.

Au final, le modèle des concepts du domaine est une spécification UML au format XML : les concepts sont décrits en UML (avec l'éditeur ArgoUML [ArgoUML]) ; ces descriptions sont ensuite traduites en structures XML.

Alors que pour les concepts nous avons réutilisé des formalismes existants, pour les autres modèles, nous avons pioché dans plusieurs techniques de description.

2. Tâches et Tâches&Concepts

Selon notre cadre conceptuel (voir chapitre précédent), le modèle des Tâches et le modèle des Tâches&Concepts sont distincts. Le premier est une description initiale tandis que le second est une description transitoire. Dans ARTStudio, ces deux modèles ne sont pas dissociés. Cette section traite donc de modélisation de tâches et des liaisons entre tâches et concepts. Notre analyse porte en 2.1 sur la structuration de tâche suivie en 2.2 d'une réflexion sur la notion de tâche élémentaire. Puis en 2.3, nous traitons des liens entre tâche et concepts avec la pondération de concepts pour terminer en 2.4 avec le formalisme d'expression du modèle.

2.1. STRUCTURE DES TÂCHES

Il existe de nombreuses façons de décrire la décomposition des tâches et leur enchaînement.

Concernant la décomposition de tâches, tous les formalismes s'accordent sur la notion de but et de sous-buts bien que la terminologie varie d'un modèle à l'autre [Balbo 1994].

Concernant l'enchaînement des tâches, on distingue deux approches :

- L'expression explicite des enchaînements au moyen d'opérateurs de séquence (séquence, parallélisme, entrelacement, etc.)comme dans ConcurTaskTree [Paternò et al. 1997] ou MAD [Gamboa et Scapin 1997]
- L'expression implicite des enchaînements de tâches au moyen de post et pré-conditions comme dans PROSPECT [Brisson et Andre 94] ou Diane^{+}.

Notons que cette dichotomie est simpliste : [Tarby 1993] montre la possibilité de passer d'une modélisation Diane^{+} en une modélisation avec des relations d'enchaînement ; et MAD inclut aussi des post et pré-conditions.

Nous optons pour une expression explicite des enchaînements qui, selon nous, facilite la projection de l'arbre de tâches en un contrôleur de dialogue.

2.2. TÂCHES ÉLÉMENTAIRES

Lors de la définition d'un arbre de tâches, le bon "grain" des tâches élémentaires doit être décidé. Certains modèles, comme PROSPECT avec ses notions d'objectifs, de but, sous-but et tâche finale, fixent quatre niveaux d'affinement tous indépendants des actions physiques de l'utilisateur sur les dispositifs d'entrée. D'autres modèles comme UAN (*User Action Notation*) [Hix et Hartson 1993] et GOMS avec le niveau KLM [Card 1983], descendent jusqu'à la tâche physique d'interaction.

Avec notre objectif de spécification indépendante de la plate-forme cible, il convient que les feuilles des arbres de tâches soient indépendantes des actions physiques. Ces feuilles constituent les tâches élémentaires de notre arbre de décomposition.

Définition. Une **tâche élémentaire** est la plus petite tâche d'interaction indépendante de la plate-forme. Si elle devait être décomposée plus avant, on obtiendrait des tâches d'interaction dépendantes de la cible.

Ainsi nous faisons l'hypothèse que toute tâche que l'utilisateur désire réaliser avec l'interface de son application, se décompose en un enchaînement de tâches élémentaires. Nous distinguons deux grandes catégories de telles tâches : Les tâches d'utilité publique et les tâches métier qui, elles, sont dépendantes du domaine.

Nous proposons quatre types de tâches d'utilité publique identifiés à partir des interacteurs des boîtes à outils graphiques usuelles. Il s'agit de :

- sélectionner,
- spécifier,
- activer,
- consulter.

Cette catégorisation appelle les remarques suivantes :

- Les tâches d'utilité publique ne sont pas nécessairement mutuellement exclusives. Par exemple, sélectionner est un sous-cas de spécifier. Mais nous préférons dire sélectionner pour un menu déroulant ou une liste, et spécifier pour un champ texte. En suivant le précepte "qui peut le plus peut le moins", un champ texte permettra de faire de la sélection mais au prix d'une perte importante de proactivité (cf. chapitre "Génération et Inférence dans ARTStudio" à la page 142).
- L'ensemble des tâches d'utilité publique ne prétend pas être exhaustif — ce serait illusoire. Aussi, le concepteur pourra ajouter à sa convenance les tâches qu'il jugera utiles lorsqu'il rajoutera de nouveaux interacteurs permettant d'accomplir de nouvelles tâches, et notamment les tâches métier associées à des interacteurs métier.
- Enfin une tâche élémentaire doit être la plus indépendante possible du contexte d'interaction. Ainsi nous dirons Consulter plutôt que Visualiser, qui sous-entend un dispositif de sortie graphique.

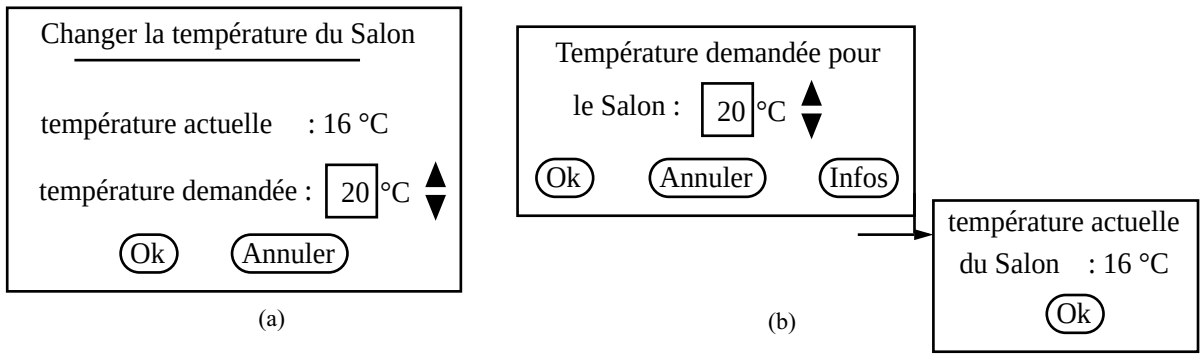
2.3. PONDÉRATION DES CONCEPTS

Lors de la description des tâches, le concepteur doit associer les concepts manipulés par chaque tâche. Selon la tâche, certains concepts sont indispensables, d'autres sont simplement utiles.

Prenons l'exemple d'un gestionnaire d'énergie de domicile et considérons la tâche "changer la température d'une pièce" (cf. figure 6). Pour cette tâche, le nom de la pièce et la température demandée sont de première importance tandis que la température actuelle est utile, mais pas indispensable. Selon la surface écran disponible (une contrainte de ressources interactionnelles dépendante de la plate-forme cible), l'IHM rendra directement observables toutes les informations nécessaires et utiles à l'accomplissement de la tâche (cas a) ou bien ne seront observables que les concepts nécessaires, les autres étant inspectables (par exemple, via le bouton "Infos" du cas b).

L'expression des pondérations par le concepteur permet, pour une tâche donnée, la distinction entre concept observable et concept inspectable.

Figure 6. IHM possibles en fonction de la pondération des concepts du domaine. Selon le niveau de pondération et les contraintes (surface d'affichage disponible) un concept de moindre importance est directement accessible si la surface écran est suffisante (a) ou accessible via une tâche articulatoire (b).



Dans une optique de génération multicible, la pondération peut changer. Par exemple, en fonction du lieu de la console de changement de la température, il peut être utile ou nécessaire de rappeler la zone pour laquelle l'utilisateur change la température. Par exemple, s'il existe une console fixée au mur par zone, alors il est inutile de rappeler la zone, sinon c'est nécessaire. Cette description dépendante de l'environnement peut s'exprimer sous forme d'une décoration (non gérée actuellement dans ARTStudio), ou se traduire par un deuxième arbre de tâches.

Dans notre système de pondération, nous distinguons deux valeurs possibles : requis/complémentaire. Un concept est dit requis pour la tâche si celle-ci ne peut être réalisée sans lui. Un concept est dit complémentaire pour la tâche s'il n'est pas requis et qu'il apporte une aide ou des informations pour la réalisation de la tâche. Cette notion d'information requise ou complémentaire a été introduite par [Sutcliffe 1997] et [Johnson et al. 1993]. Elle est équivalente à la notion d'opération obligatoire et facultative dans Diane$^+$ [Tarby 1993].

2.4. NOTRE NOTATION

Notre modèle des Tâches&Concepts s'appuie sur la notation ConcurrTaskTree [Paternò et al. 1997]. C'est un modèle des tâches augmenté des concepts manipulés dans les tâches. En l'état de l'implémentation de ARTStudio, la pondération n'est pas utilisée dans le processus de génération.

Définition d'une tâche

Une tâche est définie par :

- un nom ;
- un type (abstraction / interaction) ;
- le type de l'interaction dans le cas d'une tâche d'interaction (tâches élémentaire : consulter / spécifier / sélectionner / activer) ;
- un identificateur (déterminé automatiquement dans ARTStudio) ;
- l'identificateur de la tâche mère (défini automatiquement par ARTStudio).

Le lien avec le noyau fonctionnel est défini par :

- un prologue (actions initialisant la tâche) ;
- un épilogue (actions terminant la tâche) ;
- les concepts manipulés dans la tâche.

C'est dans le prologue et l'épilogue que sont définies les nouvelles instances des concepts et sont spécifiés les appels au noyau fonctionnel. Aujourd'hui ce lien avec le noyau fonctionnel est décrit avec le langage cible de l'application (dans notre cas Java). Les prologues et les épilogues sont de la forme :

- #NomClasse NomInstance# ;
- #NomClasse NomInstance# = *du code* ;
- *du code* ;

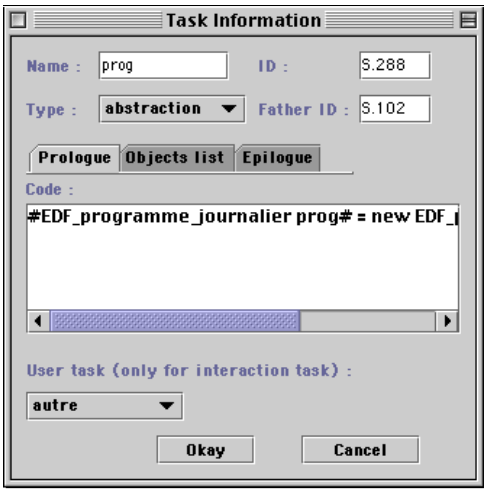
Une nouvelle instance de concept est déclarée entre '#' '#'. La figure 7 en montre un exemple.

Enchaînement des tâches

Les tâches sont reliées par des liens d'enchaînement temporel. Ce sont les opérateurs Lotos repris dans ConcurrtaskTree. Il existe différents types d'enchaînement dont l'activation avec passage de paramètre qui requiert la spécification du concept transmis en paramètre. Pour chaque lien, sont à spécifier :

- l'identificateur de la tâche de départ ;
- l'identificateur de la tâche d'arrivée ;
- le type du lien (opérateur lotos : entrelacement, choix, activation, activation avec paramètres, synchronisation et désactivation). Les

Figure 7. Interface de spécification d'une tâche dans ARTStudio.



deux derniers types ne sont pas gérés à la génération ;

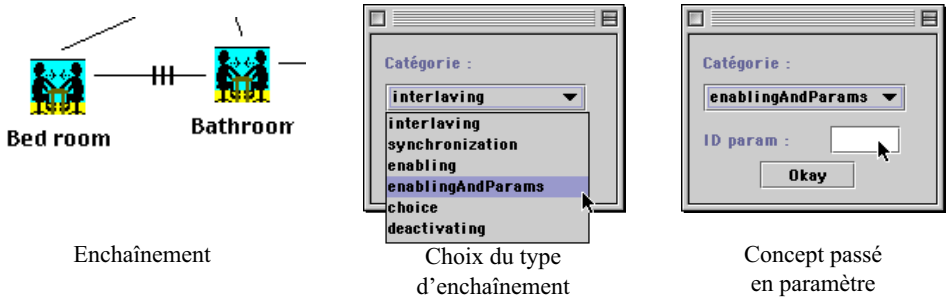
- et éventuellement l'instance du concept, qui est passée en paramètre.

Ce lien définit une relation de parenté entre deux tâches sœurs. On a évidemment :

- une tâche T et sa mère ne peuvent-être sœurs ;
- deux tâches qui n'ont pas la même mère ne peuvent être sœurs.

ARTStudio prévient les erreurs en créant les liens automatiquement. Le concepteur peut ensuite modifier le type d'enchaînement. La figure 8 montre un exemple de description de liens entre tâches.

Figure 8. Spécification de l'enchaînement de tâches.



Enchaînement

Choix du type d'enchaînement

Concept passé en paramètre

Portée des concepts Etant donné une tâche *T* et un concept *c* manipulé par T, la visibilité de *c* depuis d'autres tâches se définit comme suit :

- *c* est visible de toutes les tâches filles de *T*;
- *c* est visible de la tâche *T'* sœur de droite de *t*, si un opérateur d'enchaînement avec passage de paramètres lie les deux tâches et

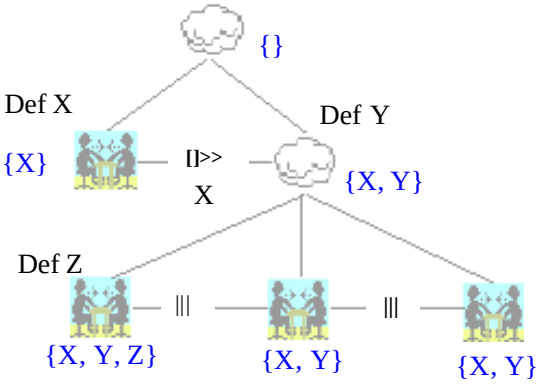
que ce paramètre est *c*.

Par récursivité sur *T'*, *c* peut-être visible dans d'autres tâches. La figure 9 illustre la portée des concepts.

Figure 9. Portée des concepts.

{} : ensemble des concepts accessibles par la tâche.

Def X : définition d'un concept X dans une tâche (dans le prologue ou l'épilogue de tâche).



3. *Interface Utilisateur Abstrait*

L'Interface Utilisateur Abstraite (IUA) décrit le squelette du dialogue. Cette structure se définit en termes d'espaces de travail, notion que l'on retrouve sous un autre vocable dans Trident et de manière implicite dans Diane^{+}.

3.1. TRIDENT

Vanderdonckt introduit deux concepts proches de la notion d'espace de travail : la fenêtre logique et l'unité de présentation dont nous recopions ici les définitions [Vanderdonckt 1997] :

Fenêtre logique : "une fenêtre logique (FL) constitue un regroupement d'informations externes manipulées par des fonctions en vue de matérialiser un contexte géographiquement délimité, nécessaire, suffisant et autonome pour accomplir un niveau quelconque de procédure d'une tâche interactive".

Unité de présentation : "une unité de présentation (UP) représente un monde d'acquisition et/ou de restitution d'information et de déclenchement de fonctions, constituée de fenêtres logiques non toutes nécessairement présentées simultanément en vue d'accomplir une sous-tâche interactive".

Ainsi les fenêtres logiques regroupent des concepts à représenter et/ou des tâches à réaliser. La fenêtre logique pourra être réalisée sous la forme d'une fenêtre à l'écran.

3.2. DIANE+ [Tarby 1993] propose une description du dialogue en buts et sous buts. Il ne définit pas explicitement le concept d'espace de travail, mais selon le niveau de décomposition, propose une méthode de choix des fenêtres et systèmes de navigation (par exemple un menu, un bouton d'activation). La génération procède comme suit :

- la fenêtre principale de l'application correspond au but principal. Toutes les autres fenêtres seront ouvertes à l'intérieur de celle-ci. Sa barre de menu est composée de menus qui portent le nom des buts de niveau 1 (buts i) et les commandes de ces menus sont composées des niveaux de buts suivants (buts i.j.k...). Les buts terminaux correspondent aux déclenchements de procédures.
- une fenêtre secondaire par procédure et par opération. Cette fenêtre aura pour titre le nom du but pour une procédure ou de l'opération pour une opération ; dans le cas où une fenêtre est issue d'une opération, son affichage dépendra de la nature de l'opération.

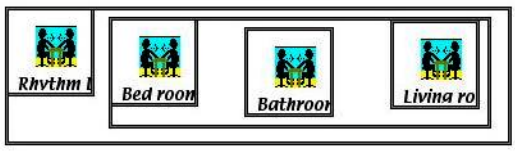
3.3. NOTRE NOTATION Comme dans Trident, nous optons pour une définition explicite de l'espace de travail. Nous le définissons comme suit :

Définition. Un **espace de travail** est une structure abstraite dans laquelle s'organise une interaction. Cette structure désigne des tâches à réaliser, des concepts à représenter, et/ou des sous espaces de travail. Ce regroupement est fait en fonction de liens entre les éléments référencés (tâches et concepts).

Un espace de travail est défini par :

- un nom ;
- un identificateur (déterminé automatiquement par ARTStudio) ;
- la tâche qu'il permet de réaliser ou une liste d'espaces de travail fils. ARTStudio ne gère pour le moment qu'une tâche par espace et un seul concept par tâche. Comme le montre la figure 10, l'espace de travail de plus haut niveau comprend deux sous-espaces. Le premier est constitué d'un seul espace pour accomplir la tâche "Changer de Rythme de vie". Le second comprend trois sous-espaces dédiés chacun à une seule tâche : "Changer la température de la Chambre", "Changer la température de la Salle de bain" et "Changer la température du séjour". Le processus de génération de cet espace à partir d'un arbre de tâches est présenté au chapitre suivant.

Figure 10. Exemple de définition d'espaces de travail.



4. Modèle des interacteurs

Le modèle des interacteurs décrit la base des interacteurs disponibles pour la production d'interfaces utilisateur concrètes (IUC). Il convient pour cela de définir cette notion de même que ses fonctions et relations au sein d'une IUC.

4.1. DÉFINITIONS

Définition. Un **interacteur** est un objet qui assure une unité d'interaction entre l'utilisateur et le système. Dans le cas d'IHM graphique, nous parlons d'"interacteur graphique" (couramment appelé widget).

Comme en système hypermédia (cf. [Kobsa et al. 2001]), nous avons identifié deux formes d'interacteurs : l'interacteur de présentation et l'interacteur de navigation.

Interacteur de présentation

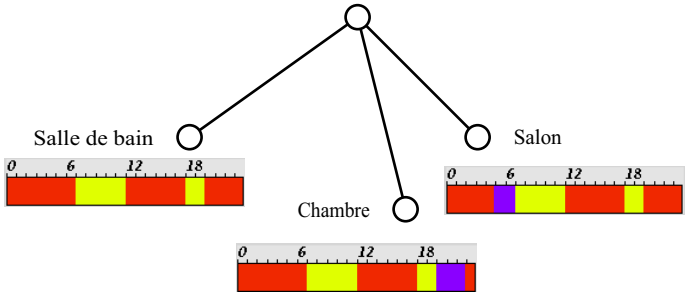
Définition. Un **interacteur de présentation** a pour fonction de représenter un concept du domaine. Il en fournit une restitution auprès de l'utilisateur et lui permet éventuellement d'accomplir des tâches de manipulation de ce concept. Par exemple, un interacteur graphique permet au moins de visualiser un concept.

Interacteur de navigation

La définition d'un interacteur de navigation s'appuie sur la définition de plusieurs notions : espace, espace navigable, outil de contrôle de navigation.

Définition. **Un espace** est un ensemble muni d'une structure.

Figure 11. Exemple d'espace. L'espace est une structure arborescente.



Définition. **Naviguer dans un espace**, c'est changer de point de vue sur cet espace (de position dans l'espace). Les techniques et les paramètres de navigation sur cet espace dépendent de la structure de l'espace.

Ainsi, naviguer dans un espace graphique revient à utiliser une technique qui permet de faire varier un point de vue visuel (par exemple au moyen d'opérations de translation, de rotation, d'homothétie).

Définition. Un **espace navigable** ou espace de navigation est un espace muni d'une fonction de navigation paramétrée.

Définition. Un **outil de contrôle de navigation** est un dispositif qui permet de jouer sur un ou plusieurs paramètres d'une fonction de navigation.

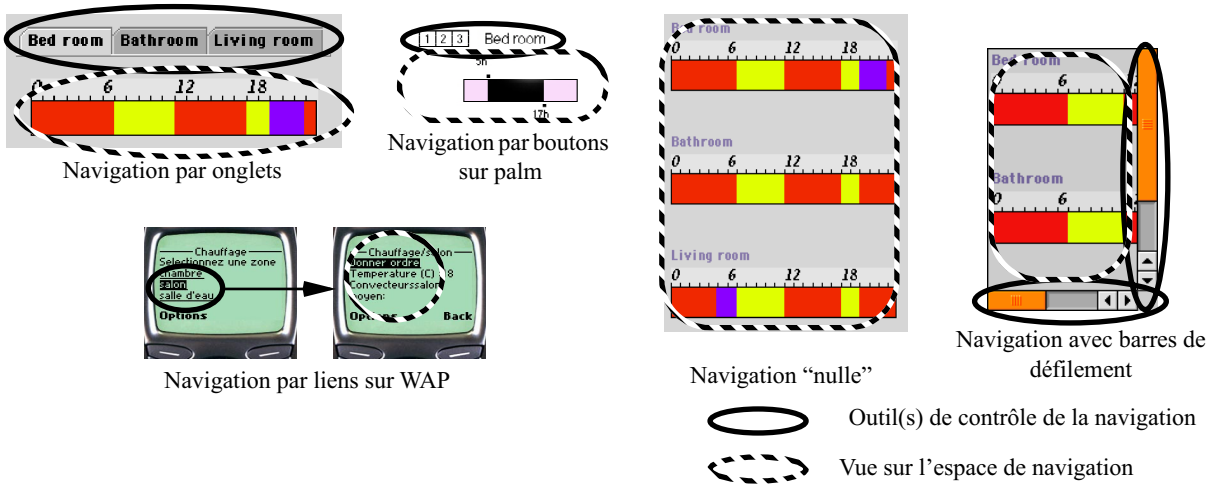
Par exemple, une manette de jeu fait varier plusieurs paramètres alors qu'un *slider* ne permet d'en manipuler qu'un seul.

Dans ARTStudio, nous nous sommes limités à la modalité de sortie visuelle, utilisant un écran comme dispositif de restitution. Dans ces conditions :

Définition. Un **interacteur de navigation** est un système composé d'une vue sur un espace navigable et de zéro, un ou plusieurs outils de contrôle de navigation sur cet espace. Le cas pour lequel il n'y a pas de contrôle de la navigation est unique à notre sens. Il s'agit de la **navigation nulle** : toutes les informations sont représentées dans la même vue. (cf. Exemples d'interacteurs de navigation figure 12).

Ayant défini nos deux catégories d'interacteur, nous abordons ses deux facettes fonctionnelle et architecturale.

Figure 12. Exemples d'interacteurs de navigation pour différentes plates-formes.



4.2. POINT DE VUE FONCTIONNEL

Il existe de nombreux travaux sur la modélisation fonctionnelle des dispositifs d'interaction. Faisons un rapide tour des approches qui ont guidé nos choix pour ARTStudio.

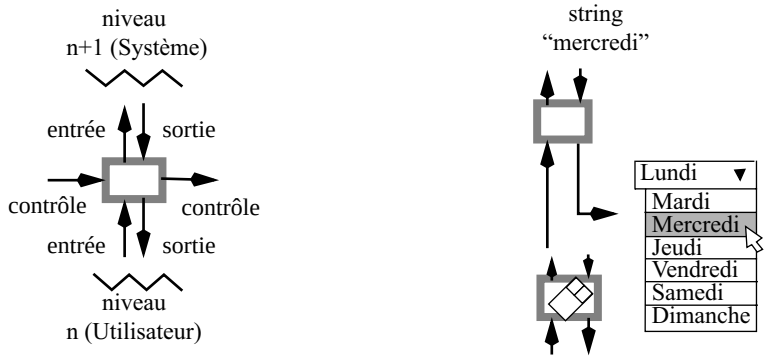
L'existant

Buxton adopte une approche centrée sur les propriétés des dispositifs physiques d'entrée [Buxton 1983]. De manière complémentaire, Foley

propose un espace fondé sur une catégorisation des tâches d'édition graphique [Foley 1984]. Avec les travaux de Mackinlay et al. nous disposons d'une première généralisation susceptible de servir la description des interacteurs avec des capacités d'entrée, de sortie, de traitement, etc.[Mackinlay 1990]. Le modèle ADC de Markopoulos va plus loin.

Avec ADC (*Abstraction-Display-Controller*), [Markopoulos 1995] décrit un interacteur comme une boîte noire qui gère des flux d'entrée, de sortie et de contrôle (cf. figure 13). Nous utilisons un modèle simplifié de cette proposition pour décrire l'interface d'un interacteur.

Figure 13. Modélisation d'un interacteur selon Markopoulos [Markopoulos 1995].

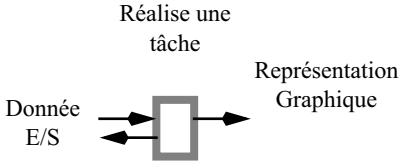


Notre proposition Un interacteur est un système capable de représenter une information auprès de l'utilisateur et lui offre éventuellement des tâches de manipulation de cette information.

En l'état, nous nous limitons au cas des interacteurs graphiques. Aussi nous modélisons un interacteur (graphique) comme suit :

- Il manipule une donnée. Cette donnée correspond à l'interface d'un flux bidirectionnel (entrée et sortie) ;
- Il a obligatoirement une représentation graphique — nous imposons un retour d'information (*Feedback*) graphique à l'utilisateur ;
- Il permet la réalisation d'une tâche. Selon la tâche, nous savons si l'interacteur est d'entrée (ex : sélectionner) ou de sortie (ex : consulter).

Figure 14. Modélisation de l'interface foncitonnelle d'interacteur.



Dans ARTStudio, un interacteur est décrit comme suit :

- son nom ;
- la liste des données qu'il est capable de représenter ;
- la liste des tâches qu'il propose ;
- son empreinte graphique (la surface d'affichage que sa représentation graphique nécessite).

4.3. POINT DE VUE ARCHITECTURE

Nous avons défini deux types d'interacteur : l'interacteur de présentation et l'interacteur de navigation. Si un interacteur de présentation peut être assimilé à un rectangle, un interacteur de navigation nécessite une structure graphique plus complexe.

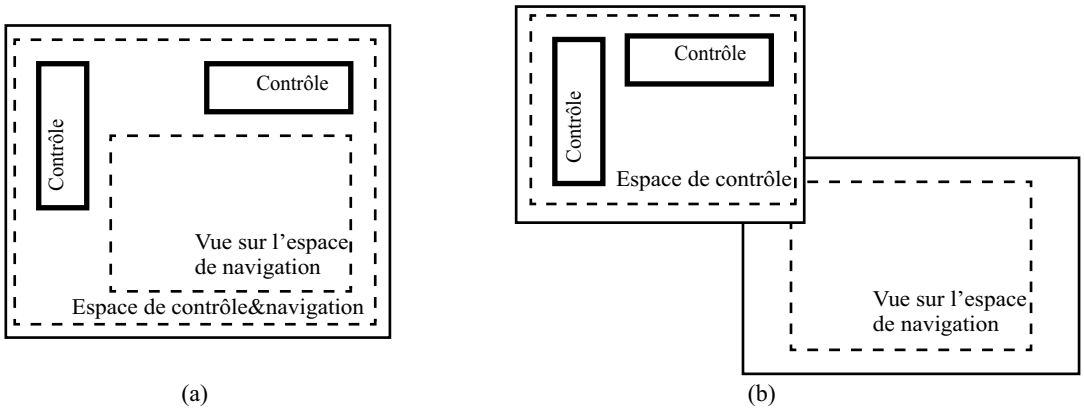
Structure graphique

Un interacteur de navigation est composé, on le rappelle, de plusieurs outils de contrôle et d'une vue graphique sur un espace de navigation. Nous énumérons deux cas possibles de structures graphiques répondant à cette définition.

- La première (cf. figure 15.a) impose que l'ensemble des objets graphiques qui constitue l'interacteur (outils de navigation et vue sur l'espace navigable) soit inclus dans un espace graphique commun.
- La deuxième structure (cf. figure 15.b) impose que les outils de contrôle soient dans un même espace graphique tandis que la vue sur l'espace navigable soit dans un espace graphique distinct.

Bien que restrictif, ce système couvre néanmoins une grande partie des systèmes de navigation que l'on rencontre dans les boîtes à outils graphiques.

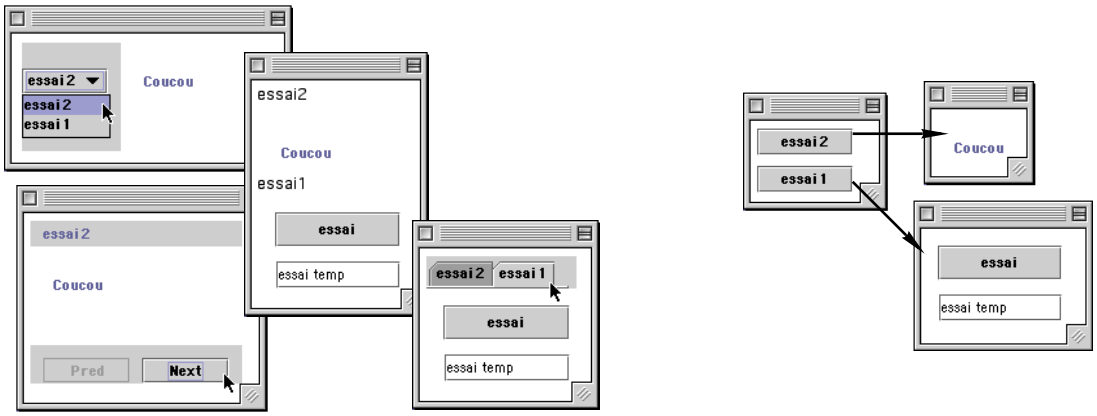
Figure 15. Les deux structures graphiques pour un système de navigation.



La figure 16 montre des réalisations possibles des deux structures. Pour le cas (a) les outils de contrôle de la navigation sont un menu déroulant, des boutons "précédent" et "suivant", des onglets ou une navigation nulle. Dans le cas (b) il s'agit de boutons ouvrant des fenê-

tres. Pour le cas particulier de la navigation nulle, donc sans outil de contrôle de navigation, la surface utilisée est celle de la vue qui couvre tout l'espace navigable.

Figure 16. Exemple de changement de contrôle de navigation. Application des deux structures graphi-



(a) (b)

A partir de cette structure, il est aisé d'intervenir sur les outils de contrôle et/ou de commuter entre la première ou la deuxième structure graphique pour faire de la plasticité. La figure 16 montre des exemples de plasticité d'interface par changement d'outils de navigation. Nous allons voir dans le paragraphe suivant comment organiser le squelette de l'Interface Utilisateur Concrète pour faciliter ce type de décomposition graphique.

Structure logicielle

La structure logicielle d'un interacteur doit permettre de générer l'Interface Utilisateur Finale de manière systématique et d'en faciliter la modification (par exemple remplacer un interacteur de navigation par un autre) sans changer la décomposition structurelle de l'architecture logicielle. Ces requis, nous ont amenés à opter pour un modèle architectural à agents.

Les modèles d'architecture conceptuelle à agents comme MVC ou PAC [Coutaz 1990] nous ont servi de référence. Les principes en sont les suivants :

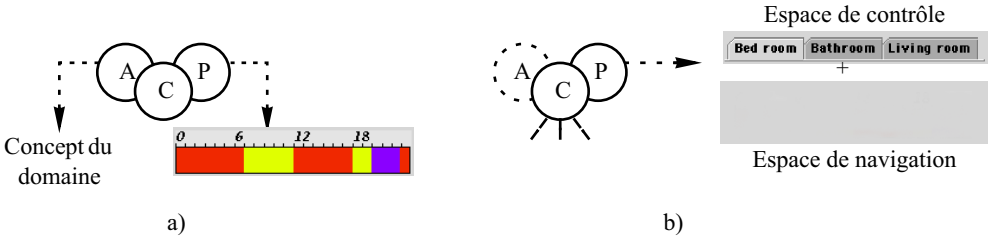
- Décomposer l'IHM en composants fonctionnels indépendants mais communicants (les agents) ;
- Distinguer pour chaque agent le comportement interne, le comportement externe, et la communication.

Comme le montre la figure 17, nous avons utilisé PAC comme fondement directeur à la modélisation architecturale de nos interacteurs.

L'agent interacteur de navigation doit respecter les points suivants :

- sa présentation inclut l'espace des outils de contrôle et l'espace de navigation ;
- il possède au moins un fils de type présentation ou navigation. Le choix de l'affichage des fils est à sa charge. Selon le type de navigation, les fils peuvent être tous affichés (navigation nulle), un seul peut être affiché (navigation par onglets), etc. La présentation des fils est affichée dans l'espace de navigation.

Figure 17. Représentation de l'architecture logicielle d'un interacteur. a) de présentation, b) de navigation.



Maintenant que nous avons défini un interacteur, son interface fonctionnelle, et son architecture (graphique et logicielle), voyons la modélisation de l'Interface Utilisateur Concrète.

5. *Interface Utilisateur Concrète (IUC)*

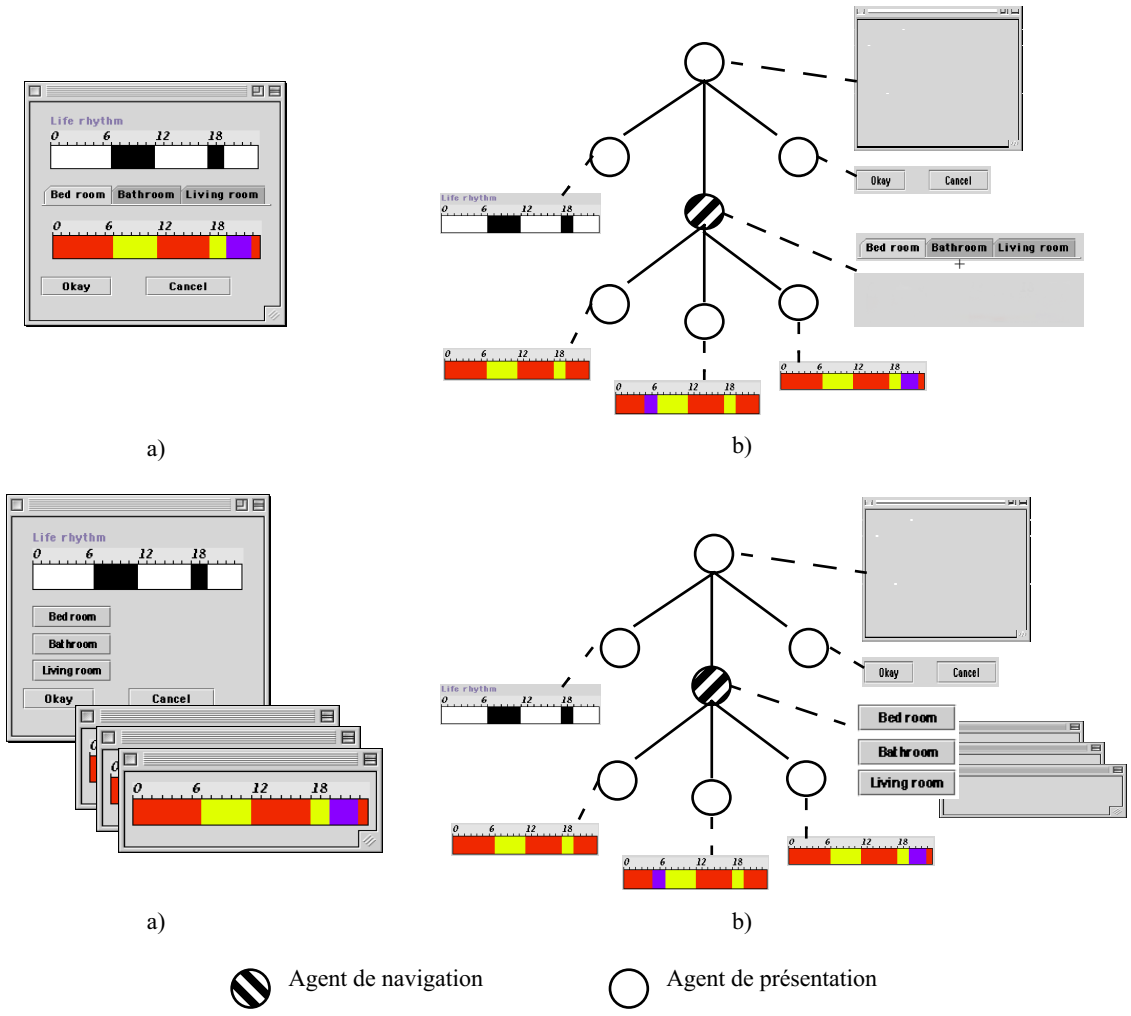
L'Interface Utilisateur Concrète (IUC) est une réalisation de l'IUA pour une cible donnée. A ce niveau de réification, les interacteurs, la navigation dans l'interface et le placement des objets d'interaction graphique (interacteurs de navigation et interacteur de "présentation") ont été définis . Faisons un zoom sur la structure de IUC et les stratégies de placement des objets d'interaction.

5.1. Structures dans l'IUC

L'IUC est le premier prototype de l'interface utilisateur finale. Elle est dépendante de la plate-forme cible, et notamment, dans le cas d'ARTStudio, de la surface d'affichage disponible et des interacteurs graphiques disponibles. La mise en œuvre d'un algorithme de génération de l'IUC se fonde entre autres sur la structure graphique d'un interacteur de présentation et de navigation.

Le squelette de l'IUC décrit le dialogue sous la forme d'une hiérarchie d'agents (à la manière de PAC [Coutaz 1990]). Nous utilisons la décomposition d'un agent (cf. section 4, "Modèle des interacteurs") pour construire ce squelette. La figure 18 présente un exemple de cette structure.

Figure 18. Structure hiérarchique de l'IUC en agents. a) est l'interface décrite, b) est le squelette de cette interface.



5.2. STRATÉGIE DE PLACEMENT

La définition de stratégies de placement, problématique issue du domaine de la mise en page, a été largement traitée. Aujourd'hui, les boîtes à outils graphiques comme Tk ou Swing/AWT, sont proposées avec différents gestionnaires de placement (*Layout manager*) implémentant des propriétés de placement prédéfinies (*CardLayout*, *FlowLaout*, etc.).

Il existe différents niveaux d'abstraction pour ces gestionnaires de placement que nous classons en niveaux lexical, syntaxique et sémantique :

- Placement lexical. Positionnement des objets graphiques selon des coordonnées cartésiennes. L'interacteur X est positionné en (10, 10) ;
- Placement syntaxique. Placement qui vérifie une structure entre les interacteurs en définissant des relations entre objets graphiques. L'interacteur X est à droite de l'interacteur Y. La structure peut-être

prédéfinie, comme dans la plupart des *Layout manager*, ou spécifiée à l'aide d'un langage de contraintes (Garnet [Myers et al. 1990], Scwm [Badros et al. 2000]);

- Placement sémantique. Placement qui vérifie des propriétés de l'interface. Ces types de placement vérifient, par exemple, l'équilibre de l'interface, l'uniformité de la densité (le gris typologique en mise en page), ou encore optimiseront l'interaction [Sears 1993] en établissant des contraintes fondées sur l'optimisation des trajectoires d'interaction.

Dans la problématique de la plasticité et de l'optimisation du placement des objets graphiques à l'écran, nous retenons les systèmes de placement syntaxique en supposant que des stratégies du niveau sémantique auront pu être traduites en règles de placement syntaxique. Le moteur de plasticité n'aura qu'à résoudre le système d'équations défini. Par rapport à cet objectif, nous sommes partis sur une approche d'expression de contraintes géométriques.

Différents types de contraintes de placement

[Vanderdonckt 1997][pp. 209] énumère un ensemble de 19 relations de placement entre objets graphiques. Nous n'allons pas toutes les rappeler ici, mais seulement les caractériser et choisir un sous-ensemble représentatif pour notre problème. Nous caractérisons la liste proposée par Vanderdonckt selon trois types de relation :

- la relation de composition. Il s'agit d'inclusion d'interacteurs ;
- la relation de positionnement relatif à un autre interacteur. Interacteur X se trouve à gauche de interacteur Y. Interacteur X se trouve à une distance de x centimètres par rapport à Y.
- la relation de positionnement par rapport à un guide. Interacteur X est justifié à gauche selon le guide G.

[Badros et al. 2000] proposent un ensemble de huit relations pour leur gestionnaire de fenêtres Scwm. Les éléments pour la mise en œuvre de ce gestionnaire sont les suivants :

- les guides (auxquels sont attachés les fenêtres, définissant des limites)
- les positionnements relatifs (par exemple "à droite de")
- les relations de distance (tel objet est à telle distance, tel objet a telle largeur (la relation peut-être égalité ou inégalité))
- les ancres. Positionnement strict sur la surface.

De ces relations, nous définissons les objets suivants (qui se traduisent en un ensemble de contraintes numériques) :

- l'objet Rectangle. Représente l'enveloppe englobante d'un interac-

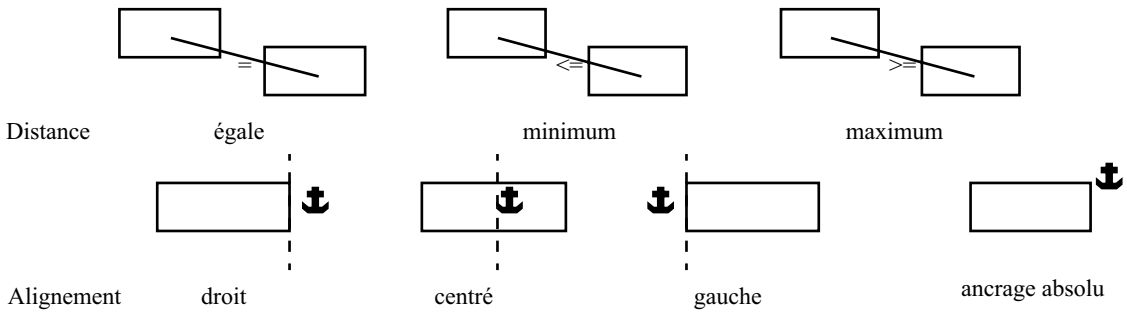
teur ;

- l'objet Guide. Représente des lignes verticales ou horizontales sur lesquelles seront ancrés (alignés) des interacteurs ;
- l'objet rectangle englobant. C'est une structure rassemblant un ensemble d'interacteurs.

et les contraintes suivantes :

- le non recouvrement. Deux interacteurs ne peuvent se recouvrir ;
- la marge. Elle définit un espace autour des interacteurs. Cet espace représente la distance minimum entre interacteurs voisins ;
- la distance. Elle définit une distance entre deux objets. Cette distance peut-être définie par une inégalité (inférieure, supérieure) ou égalité. Dans le premier cas, les objets devront être distants d'au plus (respectivement d'au moins) "d unités". Dans le second cas, les objets devront être exactement distants de "d unités". La figure 19 présente des exemples de distances ;
- le positionnement relatif. Il définit une relation de position entre deux interacteurs : au-dessus, en-dessous, à droite, à gauche ;
- l'ancre. Elle fixe un objet en abscisse et ordonnée par rapport à la surface, ou fixe deux objets entre eux. La figure 19 présente des exemples d'ancrage ;

Figure 19. Exemple de contraintes géométriques entre objets graphiques. Sur la première ligne, des contraintes de distance ; sur la deuxième ligne, des contraintes de positionnement.

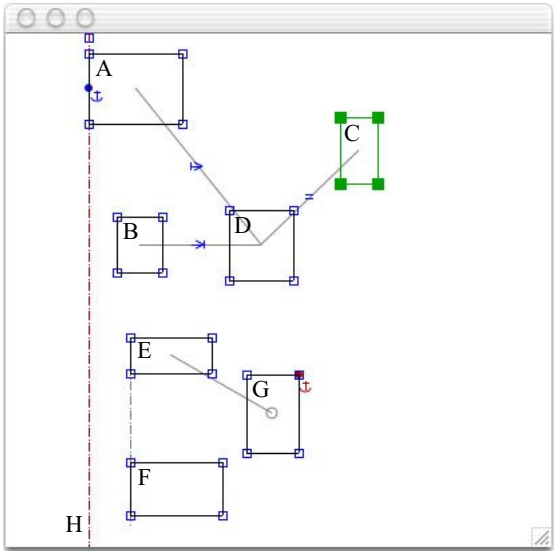


Ces objets graphiques et ces contraintes géométriques se traduisent en contraintes numériques qui seront résolues à l'aide d'un "résolveur" de contraintes numériques (cf. chapitre suivant).

La figure 20 montre la représentation des contraintes dans ARTStudio. Dans cet exemple, nous observons les interacteurs A, ..., G et le guide H, contraints comme suit :

- A est fixé à H (contrainte d'ancrage).
- A et D sont liés par une contrainte de distance minimum ;
- B et D sont liés par une contrainte de distance maximum ;
- C et D sont liés par une contrainte de distance fixe ;
- E et F sont alignés à gauche ;
- E est au-dessus de G ;
- G est positionné en absolu (contrainte d'ancrage) par un de ses sommets.

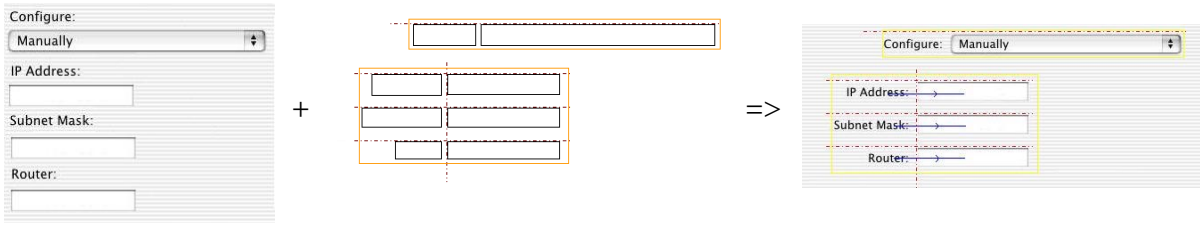
Figure 20. Exemple de contraintes de placement dans une interface. Le placement des contraintes est généré avec CUIDesigner, une extension d'ARTStudio.



Ces contraintes permettent la définition d'un modèle de présentation qui sera appliqué sur l'interface. Chaque interface adaptée respectera cette présentation dans la limite de la résolution[2] des contraintes. Aujourd'hui, ARTStudio ne génère que des interfaces avec une présentation verticale des interacteurs ; le concepteur peut ensuite la modifier manuellement. Avec l'extension CUIDesigner faite pour ARTStudio, il est possible de définir un modèle de présentation. Mais cette extension n'est pas encore intégrée dans la génération de l'IUC. La figure 21 montre un exemple d'interface générée par ARTStudio et son intégration avec un modèle de présentation.

2. Dans certain cas , le système d'équations défini par les contraintes liées au modèle de présentation et à la plate-forme, n'a pas de solution. Dans ce cas, le modèle de présentation ne peut pas être appliqué tel quel. Nous verrons cette problématique de "non solution" dans le chapitre suivant.

Figure 21. Exemple d'IUC avec un modèle de présentation.



IUC avec un modèle de présentation verticale (tel que le génère ARTStudio)

Modèles de présentation (définitsdans CUIDesigner)

IUC (visualisée dans CUIDesigner)

6. *Interface Utilisateur Finale (IUF)*

A la différence de l'IUC qui décrit un prototype d'interface (aucun lien n'est fait avec le noyau fonctionnel par exemple), l'Interface Utilisateur Finale (IUF) décrit le code de l'application. Ce code sera compilé, dans notre cas à l'aide de "javac", pour produire une interface exécutable. L'architecture implémentationnelle de ce code présentée en 6.1 s'appuie sur une architecture de référence que nous justifions ci-dessous.

6.1. ARCHITECTURE DE RÉFÉRENCE

Nous avons retenu PAC-Amodeus [Nigay 1994] comme architecture de référence. Elle autorise à la fois une décomposition fonctionnelle éprouvée (Arch) et l'affinement du contrôleur de dialogue en agents qui correspondent à nos interacteurs. Voici comment nous avons exploité cette double propriété illustrée avec la génération d'une IHM plastique pour un gestionnaire d'énergie.

Une architecture fondée sur PAC-Amodeus :

- Facilite le branchement de code pré-existant : le Noyau Fonctionnel étant fourni par ailleurs par EDF, il suffisait d'écrire un ANF sur lequel brancher le CD et la présentation décrite dans l'IUC.
- Convient à notre proposition de niveaux d'abstraction orientés système pour l'adaptation d'une interface (cf. "Constituants adaptés selon Arch" à la page 23). Cet argument prend tout son intérêt lors de la génération d'une interface faisant de l'adaptation dynamique (cf. Annexe 3 : MMS).

Enfin l'adéquation de l'architecture PAC-Amodeus a été démontrée pour les interfaces multimodales en entrée [Nigay 1994] et en sortie [Vernier 2001]. Si aujourd'hui nous ne générons que des interfaces monomodales (graphiques), en choisissant une telle architecture nous nous faciliterons l'intégration d'une génération d'interfaces multimodales dans ARTStudio.

6.2. ARCHITECTURE IMPLÉMENTATIONNELLE

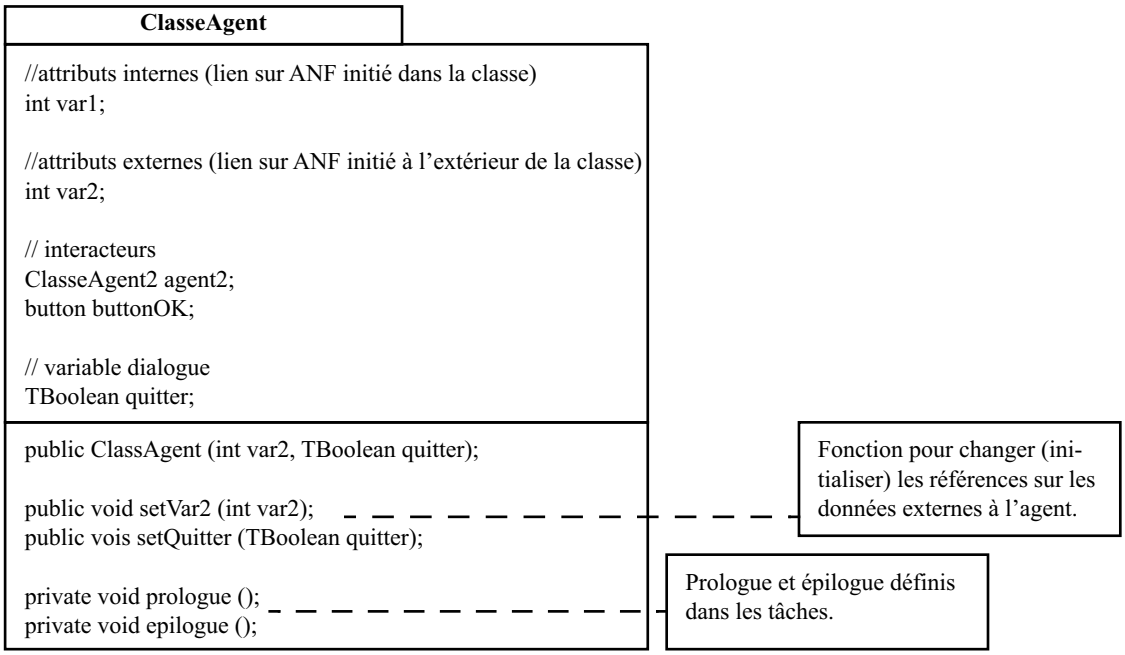
Dans notre architecture implémentationnelle, la réalisation d'un agent est faite sous la forme d'une classe Java. Cette classe possède des liens :

- sur les concepts manipulés (ANF) ;
- sur les interacteurs (P).

La communication entre les agents est réalisée à l'aide de variables partagées. La figure 22 montre une classe représentative d'un agent. Elle possède :

- des attributs internes. Ce sont des variables de l'ANF allouées par l'agent ;
- des attributs externes. Ce sont des variables de l'ANF allouées par d'autres agents et manipulées par notre agent ;
- des interacteurs. Un interacteur peut faire partie de la présentation de notre agent ou être un agent fils ;
- des variables sur le dialogue.

Figure 22. Diagramme de classes UML d'un agent.



Un point important de notre architecture porte sur la communication dans l'interface : communication entre le NF et l'ANF, entre l'ANF et le CD, au sein du CD, et entre le CD et les présentations.

Communication

L'implémentation de la communication dans une IHM est réalisée couramment selon trois façons :

- par un système de *callback*. Une *callback* est une fonction attachée à un interacteur. En général, elle est associée à un état de l'interacteur.

Lorsqu'une action est faite sur l'interacteur, la fonction correspondant à l'état obtenu par l'action, est appelée. Ainsi le dialogue peut savoir que le bouton "Okay" a été déclenché et peut agir en conséquence ;

- par un système d'événements. Lorsqu'une action est faite sur un interacteur, il déclenche un événement. Du côté du dialogue, une boucle d'acquisition des événements analyse les événements reçus et active les traitants[3] adéquats ;
- par un système de notification comme les variables actives. Un lien bidirectionnel est construit entre l'application et les interacteurs.

Bien que les variables actives n'existent pas en Java, notre langage de programmation, nous avons choisi ce modèle qui offre une parfaite indépendance fonctionnelle et simplifie l'implémentation (la génération du code) ; une grande partie de la gestion des appels est gérée dans la variable active, alors que pour un système événementiel par exemple, il est nécessaire de générer soi-même la boucle d'événements.

Nous avons implémenté ce système de variable active à tous les niveaux de l'Arche : tout objet de l'application est une instance de la classe TObject qui propose la propriété de variable active. Voyons plus précisément l'implémentation d'une variable active selon le principe de la notification.

Principe de la notification

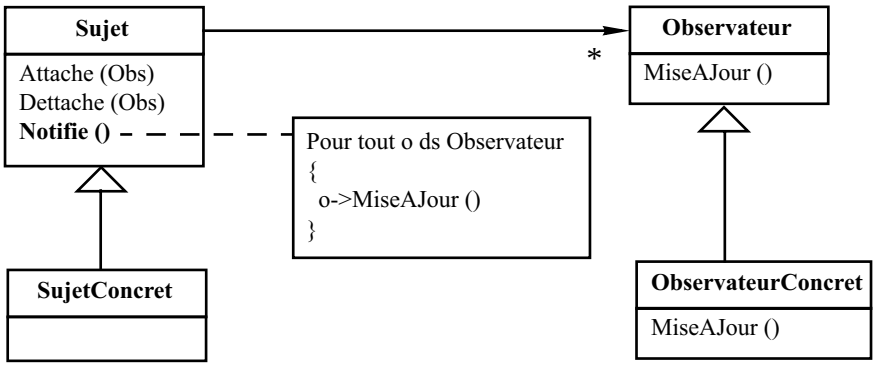
La notification est un service qui permet à un module qui le désire (l'observateur) d'être informé lorsqu'une structure de données est modifiée (le sujet) [Gamma et al. 1995]. [Fekete 1996] montre l'importance de ce principe, et la nécessité de son implémentation au niveau des données pour faciliter l'implémentation de la propriété d'honnêteté.

Le principe de notification comprend des étapes d'abonnement, de déclenchement et de traitement. En première étape, l'observateur s'abonne pour être informé d'une action en lecture ou en écriture sur un sujet. Lorsqu'une action est faite sur le sujet, la notification est déclenchée, ce qui a pour effet l'exécution d'un traitement chez l'observateur. La figure 23 présente le diagramme de classes du principe.

Le principe de notification est présent dans Java. Le langage propose en standard les classes *Observer* (l'observateur) et *Observable* (le sujet). Par contre nous l'avons développé pour la génération de code pour PalmPilot. En effet, Waba, une implémentation de Java sur PalmOS, ne fournit pas ce service.

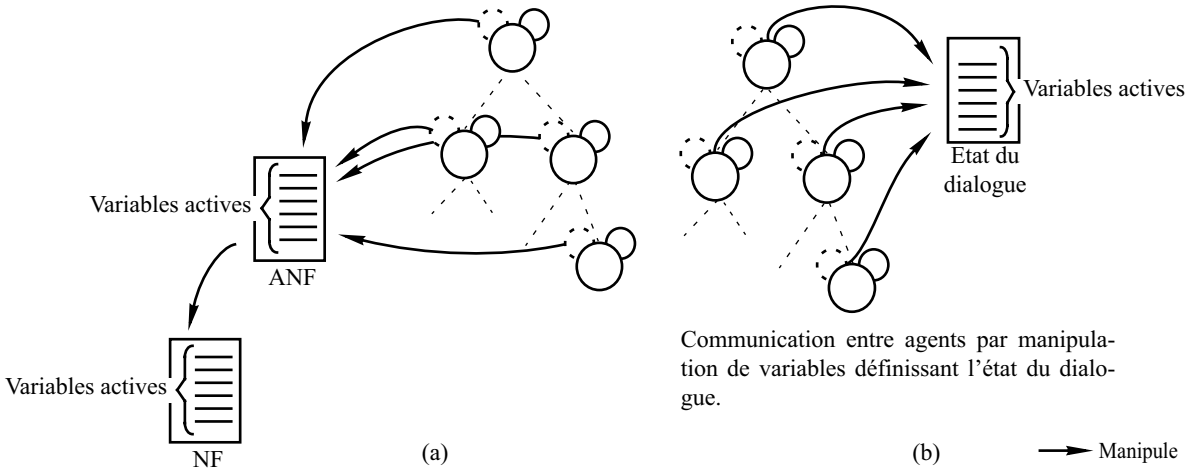
3. Fonction qui traite l'évènement en exécutant une suite d'actions.

Figure 23. Diagramme de classes UML du principe de notification. (adapté de [Gamma et al. 1995])



Implication dans PAC-Amodeus Le principe des variables actives a été implémenté à tous les niveaux de l'arche. La figure 24 montre les communications entre un agent et l'ANF, entre l'ANF et le NF et entre agents. La figure 25 montre, sous la forme d'un diagramme de classes, les relations qu'un agent peut entretenir.

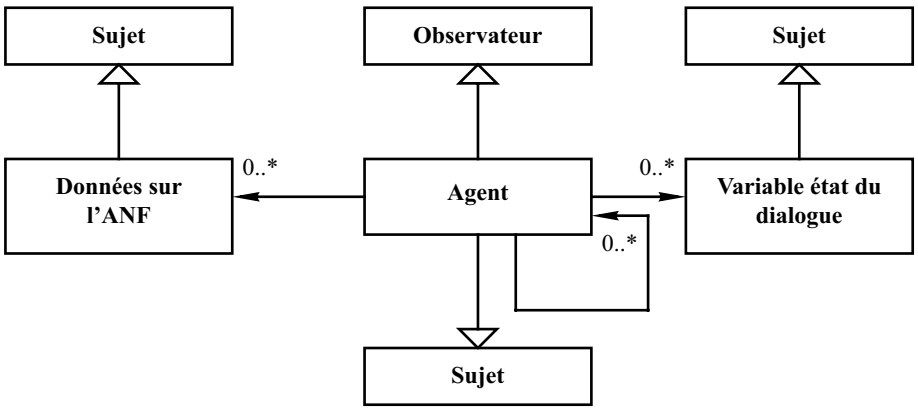
Figure 24. Liaisons dans Arch en utilisant des variables actives.



- Dans le premier cas (cf. figure 24.a), un agent, à travers son abstraction, a une connaissance de l'ANF qui lui-même a une connaissance du NF. Un changement dans le NF ou l'ANF est répercuté automatiquement à une abstraction supérieure par une notification ;
- Dans le deuxième cas (cf. figure 24.b), tous les agents pointent sur un ensemble de variables définissant l'état du dialogue. Dans ce type d'architecture, il n'y a plus de communication entre les agents. Le dialogue s'organise autour des variables actives qui définissent à un moment donné l'état du dialogue. Si l'une des variables change, alors les agents intéressés en sont avertis. Le seul moment où un agent communique directement avec un autre agent est lorsqu'un agent crée un fils : il le construit (passage en paramètre, des varia-

bles définissant l'état du dialogue et des variables sur l'ANF).

Figure 25. Diagramme de classes UML d'un agent. Il décrit les relations (héritage et association) entre un agent et l'ANF, sa présentation, ses agents fils et l'état du dialogue.



7. Résumé

En synthèse, ce chapitre a présenté l'ensemble des descriptions utilisées en pratique dans ARTStudio.

Par rapport à notre cadre de référence,

- Nous n'avons pas présenté la description de plate-forme trop élémentaire, en l'état actuel de ARTStudio : il s'agit d'une description XML qui indique les ressources interactionnelles et notamment la surface d'affichage disponible (un couple d'entier <x,y>), sa résolution (nombre de couleur), et le langage de programmation disponible (Java ou Waba).
- ARTStudio n'inclut pas de description d'environnement.

Ces limites s'expliquent aussi par la complexité du problème de la génération : inférer des solutions à partir de descriptions qui peuvent être sur-contraintes. Ces aspects sont abordés au chapitre suivant.

Chapitre 6 *Génération et Inférence dans ARTStudio*

Dans le chapitre précédent, nous avons présenté les descriptions utilisées dans ARTStudio. Nous allons voir maintenant comment les étapes de génération sont réalisées dans notre outil pour générer l'Interface Utilisateur Finale.

Ce chapitre s'organise comme suit :

- Nous rappelons en 1 nos objectifs de génération qui conduisent naturellement à une approche de génération par inférence ;
- En 2, nous résumons l'apport de quelques systèmes d'inférence pour la génération (résolution de contraintes) ;
- Puis, pour chaque étape du processus de réification, nous présentons les méthodes et les algorithmes permettant la production des descriptions transitoires et terminale de ARTStudio : Interface Utilisateur Abstraite (section 3), Interface Utilisateur Concrète (section 4), et Interface Utilisateur Finale (section 5).

Dans nos exemples, nous utilisons des pseudo-diagrammes de classes UML et des schémas XML. Nous présentons les algorithmes sous la forme assertionnelle de JESS [JESS] ou sous une forme combinée de pseudocode et d'assertions.

1. Objectifs de génération et approche

Dans le cadre de notre recherche, l'objectif de la génération, est de produire **automatiquement**[1] des IHM multicibles à partir de descriptions initiales comme les modèles de tâches et les concepts du domaine. Ces interfaces, produites selon des **règles** de conception pré-établies,

1. Le concepteur, rappelons-le, peut intervenir après chaque étape de génération (voir chapitre 4).

devront respecter des **contraintes**. Précisons nos objectifs et, en réponse à ces objectifs, l'approche générale adoptée.

1.1. OBJECTIFS

Rappelons brièvement les actions de génération :

- construire l'IUA à partir de la structure de l'arbre de tâches, la structure des concepts n'étant pas encore utilisée à cette étape ;
- choisir les interacteurs en fonction du concept à représenter et de la tâche à réaliser ;
- construire la navigation dans le dialogue en fonction de la structure de l'IUA ;
- choisir les interacteurs graphiques en fonction de contraintes de placement et de la surface d'affichage disponible pour aboutir à l'IUC.

Ces objectifs révèlent deux classes de problèmes selon la nature des contraintes à satisfaire :

- Dans les trois premières étapes, le raisonnement s'appuie sur la structure de l'information et le domaine de valeurs des concepts du domaine. Il s'agit donc de satisfaire des contraintes structurelles logiques.
- Dans la dernière étape nous avons affaire à un raisonnement sur le placement des interacteurs graphiques. Il s'agit ici de satisfaire des contraintes de nature géométrique.

Nous assistons donc à deux classes de problèmes pouvant donner lieu à des systèmes de résolution distincts selon la manière dont les problèmes sont posés, selon les performances souhaitées, la complétude de la solution recherchée, etc.[2]

1.2. APPROCHE

Nos objectifs conduisent à considérer les systèmes de raisonnement de l'"intelligence artificielle" qui s'appuient sur des moteurs d'inférence agissant sur des bases de connaissances [Haton et al. 1991]. Une base de connaissances est généralement composée d'une base de faits et d'une base de règles. Le moteur d'inférence applique les règles sur les faits pour produire de nouveaux faits (voire de nouvelles règles). Pour nous, l'intérêt de cette approche est de nous concentrer sur l'expression du problème (définition des contraintes, des règles, etc.) plutôt que rechercher une solution (c'est-à-dire, trouver un algorithme qui réponde au problème posé).

A propos des contraintes, rappelons quelques définitions tirées de [Prolog IV] et [Charman 1995]

2. Comme nous le verrons au paragraphe 2.2, un problème géométrique peut-être résolu par un système expert, en programmation logique, par résolution numérique, etc.

Définition. Une **contrainte** est l'expression de toute relation qui lie un certain nombre d'objets qui prennent leurs valeurs dans un certain domaine. A ce titre, les équations de la physique qui imposent à certaines quantités d'être égales, les impératifs qui imposent des relations particulières aux éléments d'une machine, au déroulement d'un processus, ou qui imposent des conditions pour la réalisation d'une tâche sont des contraintes.

Définition. Un problème est dit **sous-contraint** lorsque l'ensemble des contraintes définies n'est pas suffisant pour que la résolution du système donne une unique solution.

La sous-contrainte pose le problème de l'incomplétude de la solution trouvée. Ainsi comment faire pour obtenir une autre solution? Comment faire pour obtenir toutes les solutions ?

Définition. Un problème est dit **sur-contraint** lorsqu'aucune solution ne peut satisfaire l'ensemble des contraintes posées.

Ce cas est difficile à traiter car il pose le problème de recherche des contraintes qui se contredisent : problème du *nogood* [Charman 1995][pp.17], comment trouver les *nogoods* ?

2. Techniques pour l'inférence

Nous avons identifié deux classes de problème : la résolution de contraintes sur la structure et les domaines de valeur, et la résolution de contraintes de placement géométrique. Sans viser l'exhaustivité, nous présentons les solutions courantes pour ces deux types d'exigences.

2.1. STRUCTURE

Les générateurs automatiques d'IHM graphiques de type MB-IDE se fondent très souvent sur des méthodes ad hoc pour le choix, notamment, des interacteurs. [Vanderdonckt 1997] fait une revue de différentes approches. Il identifie le type de formulation du système de choix — table de correspondance, règles de production, arbre de sélection, etc.— et la provenance du savoir utilisé — règles ergonomiques, guide de style, règles de conception, etc. Par exemple, le système SEGUIA [Bodart et Vanderdonckt 1994] utilise un arbre de correspondance construit à partir de types simples de données à représenter, de leur domaine de définition et de règles ergonomiques. En sortie, il propose un interacteur appartenant aux boîtes à outils standard (radio-bouton, liste, etc.).

Un système ad hoc est souvent peu extensible, rendant difficile l'ajout de nouvelles contraintes ou règles de production. Pour cette raison,

nous optons pour des langages spécialisés en résolution de contraintes comme Prolog ou JESS.

Prolog Prolog, inventé par Alain Colmerauer à Marseille au début des années 70, est un langage de programmation logique avec contraintes. Il en existe de nombreuses implémentations dont Prolog IV [Prolog IV] qui, tout en offrant de bonnes performances, intègre plusieurs résolveurs de contraintes numériques (pouvant couvrir notre deuxième classe de problèmes). Pour des raisons techniques (interopérabilité de Prolog et de Java sur Macintosh, notre plate-forme de développement) nous n'avons pas utilisé cette implémentation.

Il existe d'autres implémentations de Prolog qui peuvent interopérer avec Java, mais nous sommes partis sur le système expert JESS.

JESS JESS pour *Java Expert System Shell*, est la combinaison d'un système expert et d'un langage de script. Il est entièrement écrit dans le langage Java. Il permet le développement de systèmes experts fondés sur l'expression de règles et d'assertions (faits). Il utilise la même syntaxe que CLIPS [CLIPS]. Nous avons retenu JESS pour trois raisons :

- Il offre de très bonnes performances, supérieures à la plupart des systèmes experts réalisés en C. Cela tient à son moteur d'inférence fondé sur l'algorithme Rete (voir lien [JESS]) qui, par "correspondance de formes" (*pattern matching*) détermine très rapidement les règles pouvant s'appliquer sur un fait donné ;
- Il offre un couplage très fort avec Java. Il a donc été très facile de l'intégrer dans ARTStudio et de faire interopérer les deux systèmes (ARTStudio exécute des commandes sur JESS, JESS appelle des procédures d'ARTStudio, ou crée des objets Java) ;
- Il est ouvert et gratuit.

2.2. PLACEMENT

Il existe plusieurs familles de résolution de contraintes géométriques [Channac 1999][Charman 1995]. Nous rappelons simplement les deux principales approches :

- L'approche algébrique consiste en l'expression des contraintes géométriques sous forme d'équations, puis en la résolution du système d'équations. Exemple de contraintes : soient trois points $A(x_a, y_a)$, $B(x_b, y_b)$, $C(x_c, y_c)$, et les équations : $x_a - x_c = 0$, $x_b - x_a = 1$, $y_c - y_b = 0$, $y_b - y_a = 1$; Le placement des points résultera de la résolution des équations.
- L'approche déductive (ou géométrique) consiste en la manipulation symbolique des contraintes géométriques pour créer un modèle géométrique concret. Le raisonnement s'appuie sur des règles géométriques (ex : AB $\perp$ BC et BC $\perp$ CD => AB // CD) pour la construction de la figure. Par exemple nous aurons des règles du

type : soient trois points A, B, C, tq, AC ⊥ CB, d(AB) = d(CB) = 1, etc.

Dans la première famille, le système d'équations est mis sous forme triangulaire, puis résolu numériquement (voir travaux de [Badros 2000]). Dans la seconde famille, la résolution du problème s'appuie sur des raisonnements faits à l'aide de systèmes experts, de langage de programmation logique, etc. (voir travaux de [Channac 1999]).

Nous sommes partis sur une approche algébrique avec le résolveur de contraintes numériques Cassowary qui s'appuie sur une extension de l'algorithme du simplex : le simplex incrémental [Badros 2000]. Ce moteur offre les avantages suivants :

- Il permet une modification interactive des contraintes, puisqu'il utilise une approche par résolution incrémentale (ou propagation de contraintes). A chaque nouvelle contrainte, le système vérifie la validité du système. Ainsi le concepteur peut modifier le modèle de présentation, qui est vérifié dynamiquement par Cassowary ;
- Il est simple à mettre en oeuvre ;
- Il en existe une version Java.

Inversement, ce type de solveur pose le problème de l'incomplétude de la solution trouvée ; il trouve une solution satisfaisant le système d'équation, mais pas toutes les solutions. Notre problème étant sous-contraint — plusieurs placements d'interacteur peuvent être valides pour une même IHM (pour le même modèle de présentation), il existe plusieurs solutions. Or celle trouvée par le solveur peut ne pas satisfaire le concepteur. Comme le concepteur ne peut pas ou ne sait pas exprimer[3] suffisamment de contraintes pour que le solveur trouve la solution qu'il aimerait avoir, jamais elle ne lui sera proposée.

Pour répondre à ce problème, il faudrait coupler ce résolveur avec un langage qui énumérerait toutes les solutions (c'est ce qui est fait en Prolog par exemple). Dans notre cas, il aurait fallu regrouper JESS et Cassowary, ou revenir sur un système du type Prolog IV.

3. Des contraintes ergonomiques ou artistiques, un savoir faire en graphisme, ne sont pas nécessairement exprimables sous la forme d'équation.

3. Squelette du dialogue : génération de l'IUA

L'Interface Utilisateur Abstraite (IUA) est produite à partir des tâches et des concepts. Cette IHM représente le squelette du dialogue. Elle s'organise comme suit :

1. Identification des espaces de travail. Il s'agit essentiellement de déter-miner les fenêtres de même que les concepts du noyau fonctionnel qui y seront présentées. Ces espaces de travail se déduisent du modèle des tâches et des concepts du domaine. L'expérience montre que les tâches proches de la racine de l'arbre de tâches correspondent à des espaces de travail distincts ;
2. Définition de la navigation entre espaces de travail. Le graphe d'appel entre ces espaces s'appuie sur le modèle de tâches qui spécifie les relations de dépendance structurelle et d'enchaînement, ainsi que sur le modèle des concepts du domaine qui décrit les relations entre concepts.

Pour obtenir une bonne adéquation outil-concepteur, l'automatisation de ces deux étapes est complexe. L'arbre de tâches conçu par les ergonomes introduit souvent, en raison des notations, des sous-tâches artificielles, donc des étapes intermédiaires indésirables dans l'IHM. De même, des tâches qui manipulent les mêmes concepts peuvent se trouver éloignées dans l'arbre alors qu'elles devraient être rapprochées. Ou encore, une tâche utilise des concepts présents dans deux espaces de travail distincts : il convient que ces espaces soient "rapprochés". Il résulte de ces observations que le passage du modèle de tâches à une structuration en espaces de travail exige des remaniements qui relèvent d'une connaissance experte encore mal formalisée pour être automatisée.

Cette section présente les heuristiques de génération utilisées dans ARTStudio et leur codage sous la forme d'assertions et de règles pour l'inférence.

3.1. HEURISTIQUES

L'Interface Utilisateur Abstraite structure l'IHM en espaces de travail et enchaînements entre espaces. Elle est obtenue par réification selon les heuristiques suivantes :

- Association d'un espace de travail final à chaque tâche d'interac-

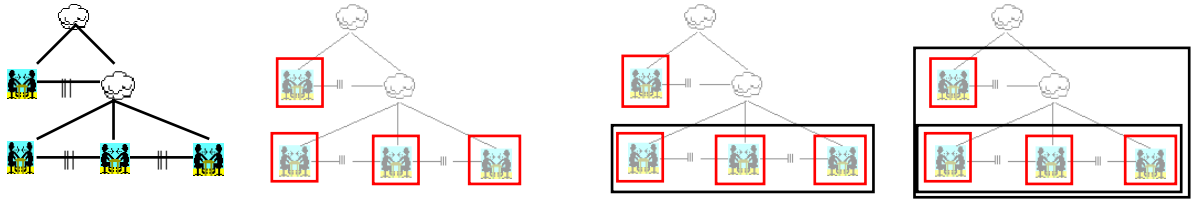
tion[4] ;

- Association d'un espace de travail à chaque tâche abstraite[5].

Les opérateurs d'enchaînement ne sont pas encore utilisés à ce stade de la réification. Ces heuristiques sont appliquées selon un processus en deux étapes (figure 1) :

1. Association des espaces de travail finaux;
2. Construction de la structure des espaces de travail en sous-espace de travail.

Figure 1. Etapes et heuristiques de génération de l'IUA à partir de l'arbre de tâches.



3.2. ASSERTIONS

Les figures 2 et 3 présentent les modèles (*template*) pour la définition des faits dans JESS. Ils décrivent les modèles de tâches, les instances de concept, et l'Interface Utilisateur Abstraite.

Figure 2. Modèles d'assertions dans JESS pour les tâches et concepts.

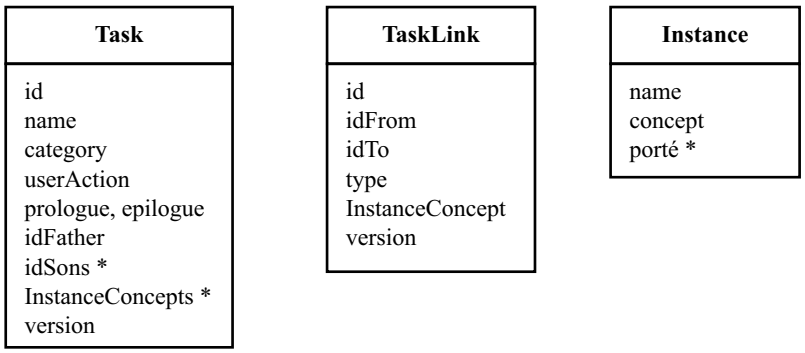
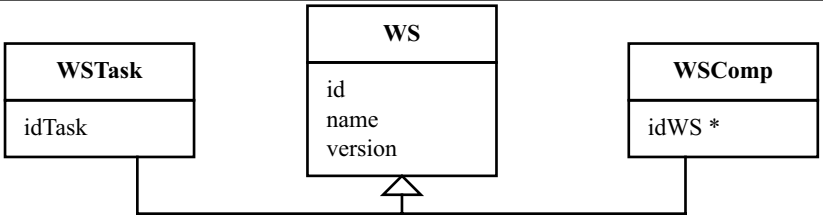


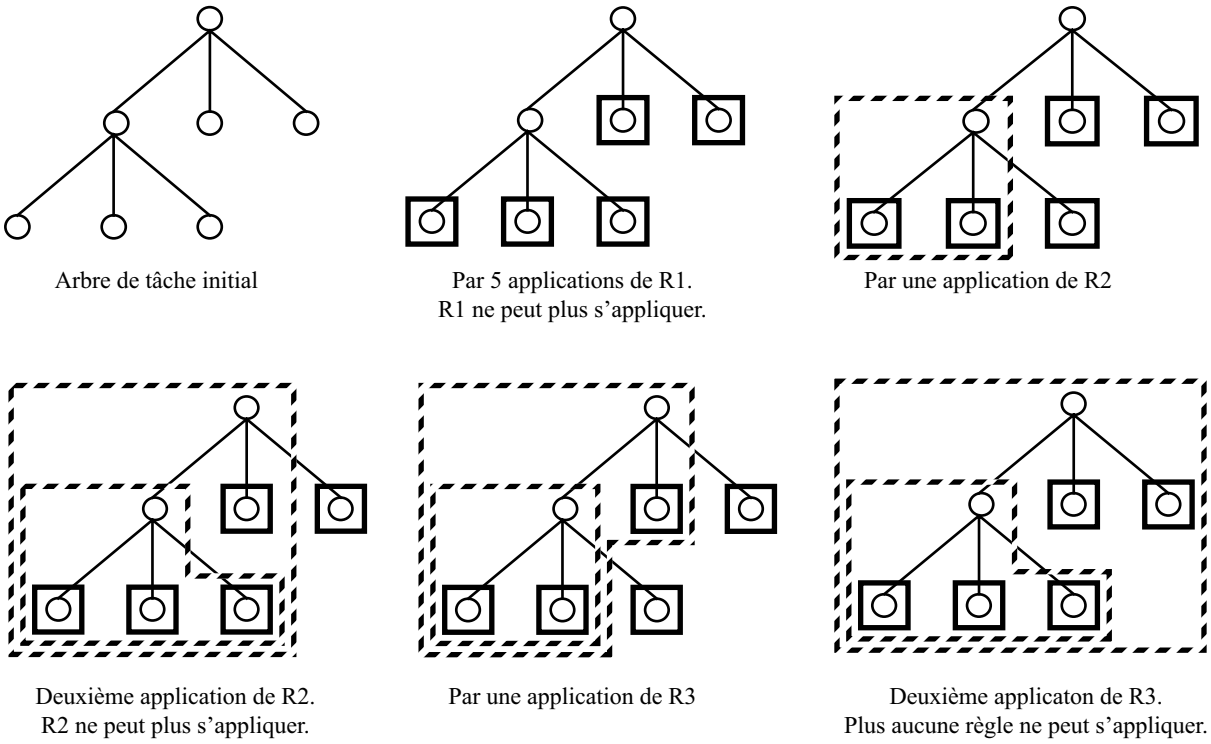
Figure 3. Modèles d'assertions dans JESS pour IUA.



4. Dans ConcurrTaskTree, formalisme retenu pour le modèle de tâches (voir chapitre 5), une tâche d'interaction est une tâche feuille dont l'accomplissement fait intervenir l'utilisateur
5. Dans ConcurrTaskTree, une tâche abstraite est un nœud de l'arbre qui regroupe un ensemble de sous-tâches.

3.3. RÈGLES La table 1 présente les trois règles de génération de IUA dans JESS. La règle R1 réalise l'étape 1 et les règles R2 et R3 correspondent à l'étape 2. La figure 4 présente un exemple d'application des règles sur un arbre de tâches.

Figure 4. Exemples de génération d'une IUA. Application sur un arbre de tâches, des régles d'inférence décrites dans la table 1.



	Règles
R1 :	(task (version ?v) (id ?id) (category "interaction") (name ?name)) => (assert (WSTask (version ?v) (id ?id) (name ?name) (idTask ?id)))
R2 :	?f <- (taskLink (version ?v) (idFrom ?idf) (idTo ?idt) (type ?t)) (task (id ?idf) (idFather ?id)) (task (id ?id) (name ?name) (version ?v)) (not (WSComp (id ?id) (version ?v))) => (assert (WSComp (version ?v) (id ?id) (name ?name) (idWS ?idf ?idt))) (retract ?f)

Table 1: Règles d'inférence pour la génération d'une Interface Utilisateur Abstraite.

	Règles
R3 :	?f1 <- (taskLink (version ?v) (idFrom ?idf) (idTo ?idt) (type ?t)) (task (id ?idf) (idFather ?id)) (task (id ?id) (name ?name) (version ?v)) ?f2 <- (WSComp (id ?id) (version ?v) (idWS $?listeID) (name ?name)) => (if (member$?idf ?listeID) then (modify ?f2 (idWS (insert$?listeID 1 ?idt))) else (modify ?f2 (idWS (insert$?listeID 1 ?idf)))) (retract ?f1)

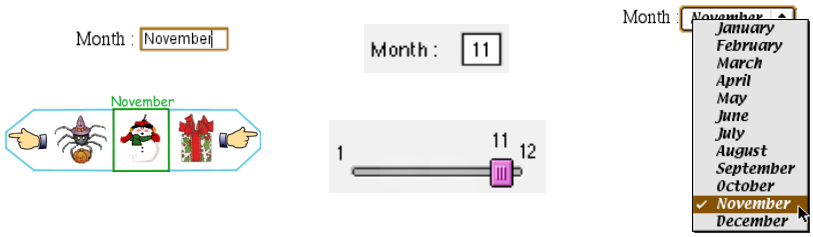
Table 1: Règles d'inférence pour la génération d'une Interface Utilisateur Abstraite.

4. Concrétisation : génération de l'IUC

Cette phase produit, à partir de l'IUA et du modèle des interacteurs, l'Interface Utilisateur Concrète (IUC). Si l'IUA était la plus indépendante possible de la cible, cette interface est complètement dépendante de la cible. L'IUC est un prototype de l'interface de l'application. Elle est produite à partir des trois étapes suivantes :

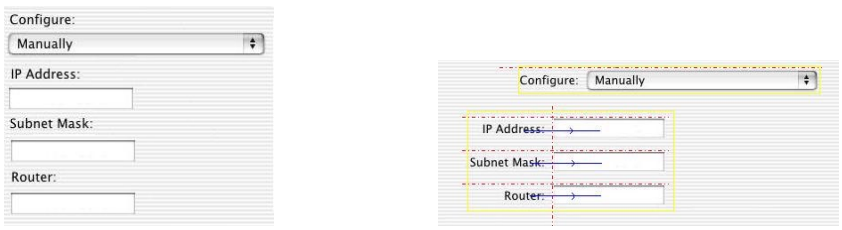
1. **Choix des interacteurs de présentation.** Il s'agit de faire une liaison entre les tâches à réaliser, les concepts du noyau fonctionnel et les interacteurs graphiques disponibles sur la plate-forme ;

Figure 5. Exemple de choix d'interacteurs de présentation. Ensemble des interacteurs permettant la tâche "spécifier un mois"



2. **Construction de l'interface.** Il s'agit de déterminer la navigation dans l'interface et d'instancier les espaces de travail sous la forme de fenêtre par exemple ;
3. **Placement des interacteurs.** Positionnement des interacteurs graphiques dans l'IUC et résolution des contraintes de placement.

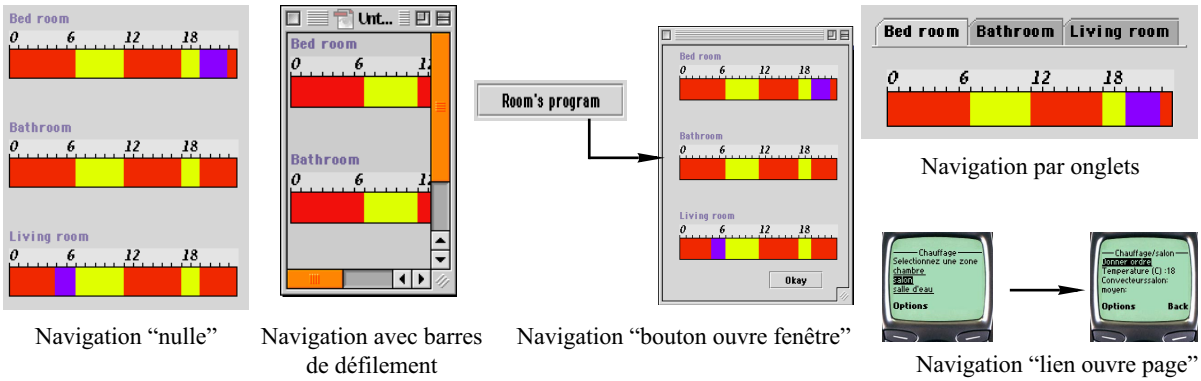
Figure 7. Exemple de placement d'interacteurs.



Placement simple (règles de positionnement vertical)

Placement avec des règles complexes

Figure 6. Exemple de construction d'une IUC. Sa structure dépend des interacteurs de navigation.



Ces trois étapes sont présentées en séquence dans les paragraphes qui suivent.

4.1. Choix des Interacteurs de Présentation

Le choix d'un interacteur de présentation dépend de ses capacités représentationnelles et fonctionnelles. Nous présentons trois algorithmes et une méthode qui favorise la proactivité :

- Le premier algorithme "DonneInteracteurs (α, *t*)" fournit tous les interacteurs capables de représenter le concept α et de proposer la tâche *t* ;
- Les deuxième et troisième algorithmes (DonneInteracteursParGeneralisation (α, *t*) et DonneInteracteurParComposition (α, *t*)utilisent la structure d'un concept complexe pour construire un interacteur ;
- Enfin nous présenterons une méthode pour augmenter la proactivité qui s'appuie sur le domaine de valeurs des concepts du domaine pour informer l'utilisateur.

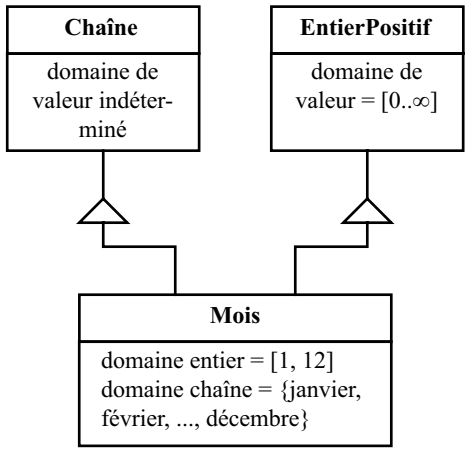
Aujourd'hui les deuxième et troisième algorithmes ne sont pas intégrés dans ARTStudio pour des raisons de combinatoire d'interface. En effet si nous devions représenter quatre concepts à l'écran et que le système trouve en moyenne trois interacteurs possibles par concept, alors nous aurions $3^4 = 81$ possibilités d'interface. L'amélioration de la proactivité n'a pas été implémentée.

Pour la compréhension de nos algorithmes, nous allons présenter la spécification des concepts de "Mois" et "Date" (cf. les figures 8 et 9).

Trouver un interacteur

La figure 10 présente les modèles d'assertion dans notre moteur d'inférence JESS.

Figure 8. Spécification du concept de "Mois". a) diagramme de classes, b) schéma XML.

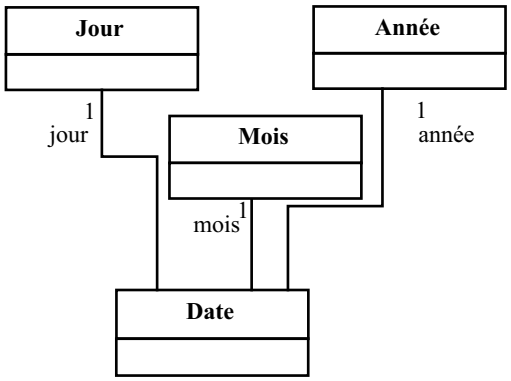


a)

```
<xsd:simpleType>
  <xsd:union>
    <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:minInclusive value="1"/>
      <xsd:maxInclusive value="12"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType>
    <xsd:restriction base="xsd:String">
      <xsd:enumeration value="janvier"/>
      <xsd:enumeration value="février"/>
      <xsd:enumeration value="mars"/>
      <xsd:enumeration value="avril"/>
      <xsd:enumeration value="mai"/>
      <xsd:enumeration value="juin"/>
      <xsd:enumeration value="juillet"/>
      <xsd:enumeration value="aout"/>
      <xsd:enumeration value="septembre"/>
      <xsd:enumeration value="ocobre"/>
      <xsd:enumeration value="novmebre"/>
      <xsd:enumeration value="décembre"/>
    </xsd:restriction>
  </xsd:simpleType>
 </xsd:union>
</xsd:simpleType>
```

b)

Figure 9. Spécification du concept de "Date". a) diagramme de classes, b) schéma XML

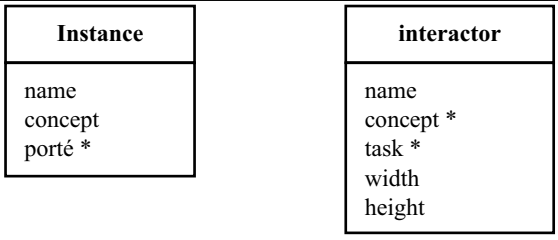


a)

```
<xsd:complexType name="Date">
  <xsd:sequence>
     <xsd:element name="jour" type="xsd:Jour"/>
     <xsd:element name="mois" type="xsd:Mois"/>
     <xsd:element name="Année" type="xsd:Année"/>
  </xsd:sequence>
</xsd:complexType>
```

b)

Figure 10. Modèles d'assertions dans JESS.

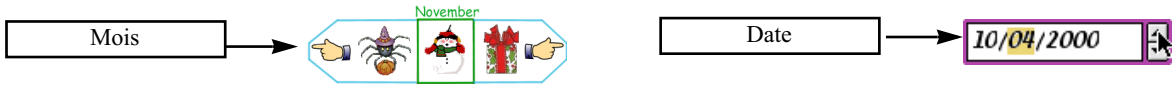


Soient α une instance de concept du domaine et *t* une tâche.
La fonction DonneInteracteurs (α, *t*) rend la liste des interacteurs en fonction de *t* et α :

```
DonneInteracteurs (α,t)
    result = {}
    pour (instance (name ?α) (concept ?c)) et
    pour tout (interactor (name ?n) (concept $? ?c $?) (task $? ?t $?))
        result = result o {n}
    retourne result
```

ARTStudio choisit le premier de la liste!

Figure 11. Exemples de choix d'interacteurs. Choix pour le concept de "mois" et de "date" et la tâche "spécifier".



Remonter l'arbre de spécialisation

Le deuxième algorithme remonte l'arbre d'héritage de l'instance α et trouve tous les interacteurs pouvant représenter des super-classes de ce concept et réaliser la tâche *t* (cf. figure 12). Les super-classes de α sont obtenues par la fonction *Isa(α)* [Ducournau 1996]. *Isa(α)* parcourt la spécification UML pour déterminer toutes les classes. Attention *Isa(α)* donne également la classe de α. Donc le résultat de ce deuxième algorithme contient le résultat de DonneInteracteurs (α, *t*).

DonneInteracteursParGénéralisation (α, *t*) :

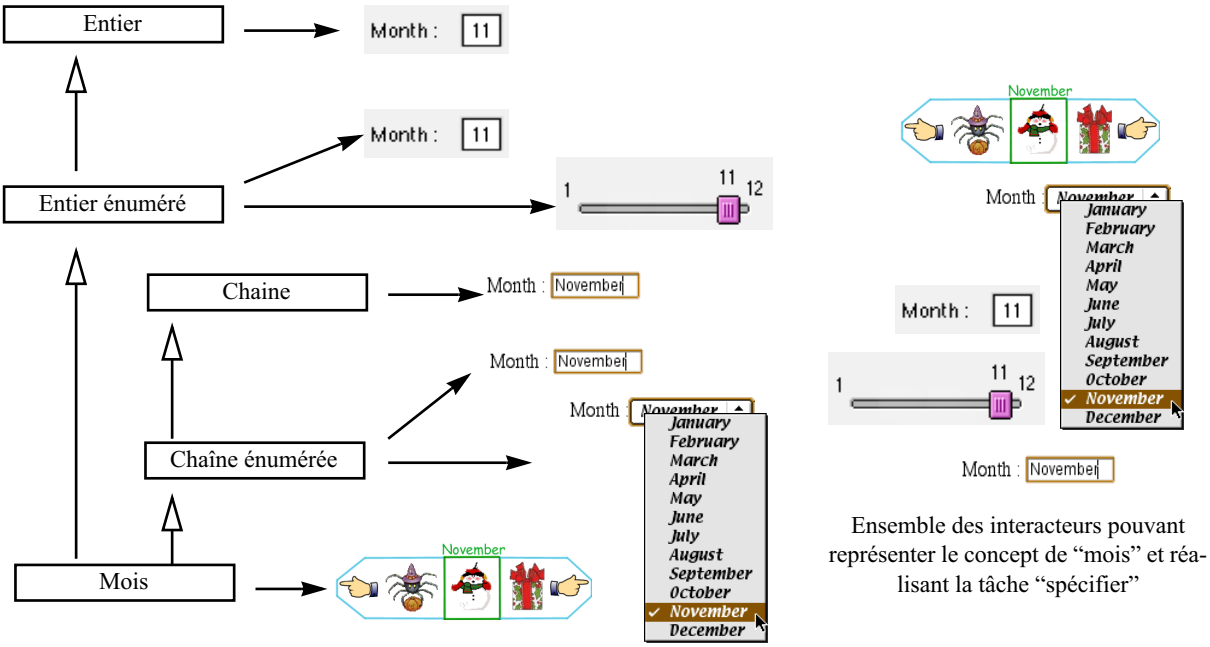
```
DonneInteracteursParGénéralisation (α, t)
    result = {}
    C = Isa (α)
    pour tout c dans C et pour tout (interactor (name ?n) (concept $? ?c $?) (task $? ?t $?))
        result = result o {n}
    retourne result
```

Remonter l'arbre de composition

Le dernier algorithme construit un interacteur en fonction de la structure du concept. Il détermine les attributs définissant le concept, trouve un interacteur représentant chaque attribut et proposant la tâche *t*, et construit le nouvel interacteur à partir de la liste des interacteurs trouvés.

Soit c une classe et P_c l'ensemble des attributs de c. P_c est calculé en parcourant la spécification UML du concept α.

Figure 12. Choix d'interacteurs par généralisation. (cf. la spécification du concept de "Mois" figure 8).



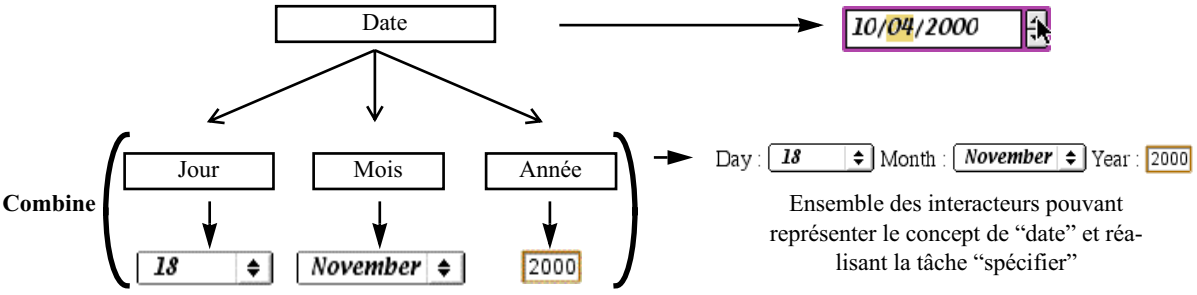
Ensemble des interacteurs pouvant représenter le concept de "mois" et réalisant la tâche "spécifier"

DonneInteracteurParComposition (α, *t*) :

DonneInteracteurParComposition (α, *t*)
result = {}
pour (instance (name ?α) (concept ?c))
pour tout *c'* dans P_c
si existe (interactor (name ?n) (concept $? ?*c'* $?) (task $? ?*t* $?))
alors result = result o {n}
sinon retourne erreur // il n'existe pas d'interacteur représentant α et proposant la tâche *t*
retourne Combine (result)

"Combine" est une fonction complexe qui construit un interacteur à partir d'une liste d'interacteurs. La figure 13 montre un exemple d'application de celle-ci. Pour déterminer les labels, il est possible d'utiliser le nom de l'attribut représenté.

Figure 13. Choix d'interacteurs par construction.



Ensemble des interacteurs pouvant représenter le concept de "date" et réalisant la tâche "spécifier"

Cette version d'algorithme est simple puisqu'elle ne cherche qu'un interacteur par attribut. Une version plus complexe trouverait tous les

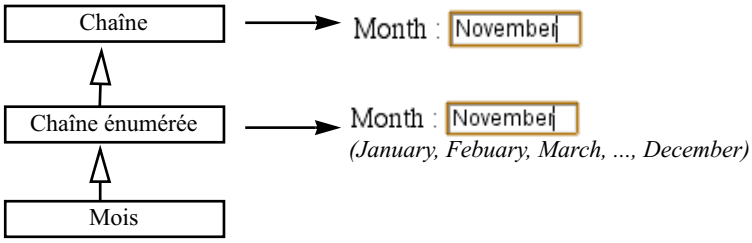
interacteurs par attributs et ferait une combinaison pour générer une liste de nouveaux interacteurs représentant α et proposant la tâche *t*. Ce problème complexe rejoins les travaux en génération d'interface à partir de données [Arens et Hovy 1995], [Brusilovsky 2001], [Roth 1990], etc.

Augmenter la proactivité

Définition. Tirée de [Calvary 1998] : "la proactivité caractérise un phénomène qui s'exerce d'amont en aval dans le temps. Elle se réfère à une progression continue et directe, prohibant tout aller-retour, source d'inefficacité [...] aider le concepteur (pour nous l'utilisateur) à progresser, de façon directe et efficace [...] si possible sans détour"

Lorsqu'un interacteur représentant un concept n'existe pas, il est possible d'en trouver un en remontant la hiérarchie des concepts (avec la fonction DonneInteracteursParGénéralisation). Or, plus l'algorithme remonte dans l'arbre de généralisation, plus les interacteurs trouvés gagneront en sémantique (cf. la figure 12). Par exemple, au lieu d'avoir un interacteur dédié pour spécifier un mois, le système proposera un "champ texte"! Pour aider l'utilisateur, il est possible d'augmenter la proactivité en utilisant le domaine de définition du concept à représenter. Ce domaine de définition est affiché sous la forme d'une aide directive. La figure 14 montre un exemple.

Figure 14. Augmenter la proactivié.



4.2. CONSTRUCTION DE L'IUC

Avant de continuer, voyons deux éléments structurant une interface : la fenêtre et le canevas.

- Une fenêtre représente un espace d'affichage à l'écran. Elle peut être manipulée par l'utilisateur (par exemple redimensionnée, déplacée, etc.). Parce qu'une fenêtre est manipulable, cet espace est généralement composé d'une bordure offrant les fonctions de manipulation (icône de miniaturisation, de fermeture, etc.).
- Un canevas (ou panel) est un espace d'affichage "nu" (il ne propose aucune fonction de manipulation auprès de l'utilisateur). Une fenêtre hérite de cet espace. Pour être utilisé, il doit être inclus dans un autre espace d'affichage (fenêtre ou canevas). Il existe un canevas (panel) initial incluant tous les autres : c'est "l'écran".

La construction de l'IUC se décompose en deux sous-problèmes liés :

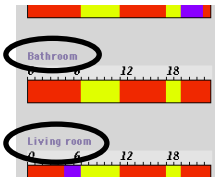
1. le choix des systèmes de navigation ;
2. l'instanciation d'un espace de travail en fenêtre ou canevas.

Choix des interacteurs de navigation

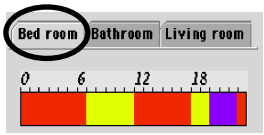
Par défaut, ARTStudio choisit une navigation par bouton (cf. figure 16). En fonction du type de navigation, il ajoute une aide. Il utilise pour cela le nom de l'instance du concept représenté (cf. figure 15) ou le nom de la tâche à réaliser (cf. figure 16).

Figure 15. Aide à la navigation utilisant les infos sur un concept. Elle peut-être incluse dans l'interacteur de contrôle, sinon elle doit être ajoutée

Sur Palm. Navigation par boutons. Ajout explicite de l'aide à la navigation

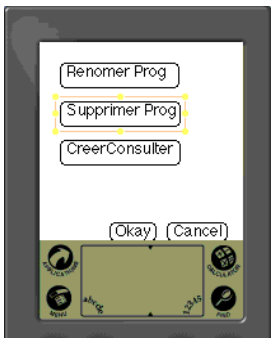


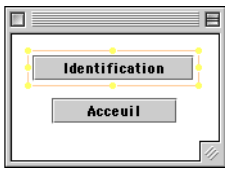
En Swing (Mac/PC), navigation "nulle". Ajout explicite de l'aide à la navigation



En Swing (Mac/PC), navigation par onglets. l'aide à la navigation est implicite.

Figure 16. Aide à la navigation utilisant les infos sur une tâche.





Structure de l'interface

Tout espace de travail est instancié par une fenêtre ou un canevas selon le choix des interacteurs de navigation. Puisqu'ARTStudio génère par défaut une navigation par bouton, tout espace de travail est instancié sous la forme d'une fenêtre graphique.

Nous avons défini les règles de génération suivantes :

- le travail fait dans un espace de travail instancié sous la forme d'une fenêtre devra est validé (ou invalidé) manuellement par l'utilisateur final ;
- le travail fait dans un espace de travail instancié sous la forme d'un

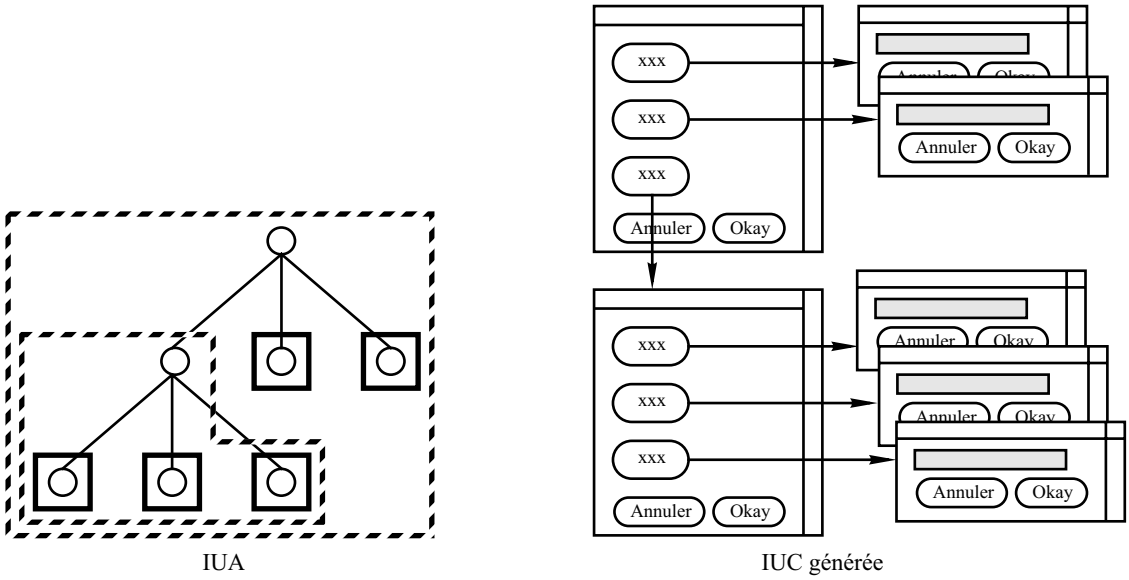
canevas sera validé automatiquement.

Ces règles impliquent les choix suivants :

- des boutons Okay/Annuler dans chaque fenêtre ;
- absence de boutons Okay/Annuler dans un canevas ;
- une fenêtre manipule un "*proxy*" de l'ANF ;
- un canevas manipule l'ANF (ou le *proxy* de sa fenêtre mère). Il n'a pas de *proxy*;

Ainsi, à partir de l'IUA présentée en figure 4, ARTStudio génère l'IUC de la figure 17. Rappelons qu'il ne s'agit pas de l'IHM exécutable mais d'un prototype d'IHM.

Figure 17. Exemples de génération de l'IUC à partir de l'IUA.



IUA IUC générée

Dans le but de proposer différentes solutions d'interface, nous avons travaillé sur un système de génération un peu plus sophistiqué développé dans MMS (cf. Annexe 3 : MMS). Il propose différentes interfaces en faisant de l'adaptation dynamiquement sur les interacteurs, et permet la vérification de la propriété de cohérence (un même concept est représenté de la même manière dans toute l'interface).

4.3. PLACEMENT DES INTERACTEURS

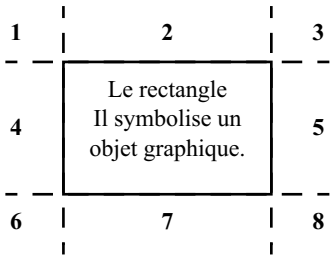
La dernière difficulté pour la production de l'IUC est la résolution du placement des interacteurs. ARTStudio comme MMS, sont très simples, et positionnent les objets graphiques suivant un placement vertical. Dans ce cas, vérifier les contraintes d'espace d'affichage revient à vérifier que la somme des hauteurs d'interacteurs (plus un espace entre chaque interacteur) est inférieure à la hauteur de la surface d'affichage disponible et que l'interacteur de largeur maximale a une largeur infé-

rieure à la largeur de la surface d'affichage disponible (aux constantes près : bordures de fenêtre, etc.).

Pour aller plus loin, nous avons travaillé sur l'élaboration d'un système de contraintes pour le placement des objets graphiques et la résolution du système défini par ces contraintes. Nous avons utilisé Cassowary qui a été intégré dans l'éditeur d'IUC : CUIDesigner d'ARTStudio. Les différentes contraintes de placement ayant été définies dans le chapitre précédent, voyons maintenant les algorithmes de résolution. Nous rappelons qu'un interacteur graphique est défini par un rectangle.

Non recouvrement

L'algorithme de non recouvrement est un méta niveau rajouté à Cassowary. Il gère pour chaque rectangle de l'interface, sa position par rapport aux autres rectangles. Huit positions sont possibles entre deux rectangles :



Soient,

- Les rectangles A et B définis par les coordonnées (x, y) de leurs quatre sommets.
- La fonction H(B, A) qui retourne la position horizontale de B, par rapport à A (droite, gauche, interzone).
- La fonction V(B, A) qui retourne la position verticale de B par rapport à A (haut, bas, interzone)

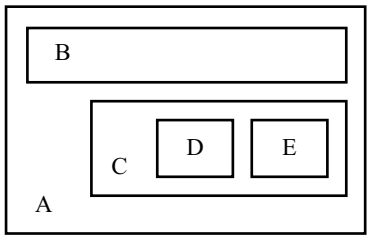
Nous définissons le méta-algorithme de non recouvrement comme suit :

Si (H(B,A) = (droite OU gauche)) ET (V(B,A) = (haut OU bas)), aucune contrainte n'est ajoutée	cas 1,3,6,8
Si (H(B,A) = droite) ET (V(B,A) = interzone), on ajoute dans le résolveur la contrainte : $x_{2A} <= x_{1B}$	cas 4
Si (H(B,A) = gauche) ET (V(B,A) = interzone), on ajoute dans le résolveur la contrainte : $x_{2B} <= x1A$	cas 5
Si (V(B,A) = haut) ET (H(B,A) = interzone), on ajoute dans le résolveur la contrainte : $y_{4B} <= y_{1A}$	cas 2
Si (V(B,A) = bas) ET (H(B,A) = interzone), on ajoute dans le résolveur la contrainte : $y_{4A} <= y_{1B}$	cas 7

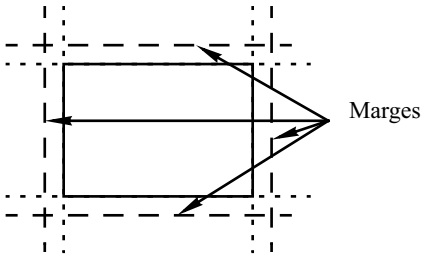
Optimisation.

Le non recouvrement est calculé pour chaque interacteur avec tous les autres interacteurs de la fenêtre. Du fait de la hiérarchisation des interacteurs, une optimisation aisée à réaliser, consiste à ne tester que le recouvrement entre interacteurs de même père. Dans l'exemple sui-

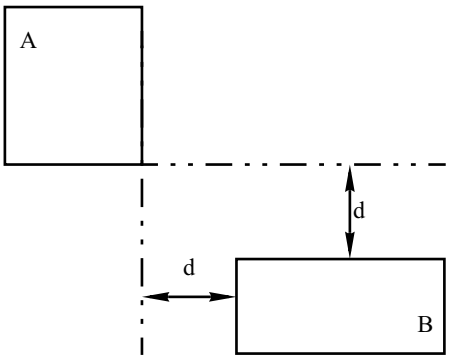
vant, il n'est nécessaire de tester le non recouvrement qu'entre D et E, ou entre C et B. Par contre il n'est pas nécessaire de vérifier le non recouvrement entre D et B puisque D est inclus dans C. Donc si B et C ne se recouvrent pas, D et B ne le peuvent.



Marge La marge est une extension du non recouvrement. Il suffit de rajouter une constante (la marge) lors du positionnement des contraintes par l'algorithme de non recouvrement.

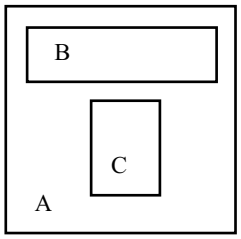


La distance C'est un abus de langage de parler de distance entre deux rectangles. En effet, une distance en mathématiques se définit entre deux points et doit vérifier quatre propriétés. Une distance entre deux rectangles ne peut pas être définie puisqu'elle contredirait certaines propriétés. Aussi nous définissons une pseudo-distance *d*(A, B) avec les contraintes suivantes (cas de l'égalité) : $x_{2A} = x_{1B} + d$, $y_{3A} = y_{1B} + d$, etc.



Rectangle englobant C'est un rectangle contenant d'autres rectangles. Par exemple un *GroupBox* contenant des *CheckBox*. Les coordonnées des sommets des rectangles englobés, doivent être à l'intérieur de l'espace défini par les

sommets du rectangle englobant. Les contraintes se posent très bien en Cassovary : par exemple $x_{1A} <= x_{1B}$, $x_{2A} >= x_{2B}$, etc.



5. Code exécutable : génération de IUF

Pour produire un code exécutable, avec une interface qui interagit non seulement avec l'utilisateur mais aussi avec le noyau fonctionnel, il est nécessaire de travailler sur les règles de génération qui devront être appliquées pour structurer le code et obtenir une application exécutable. Cette section comprend deux sous-parties :

- les principes généraux de l'architecture logicielle ;
- les principes détaillés des règles de génération de ARTStudio.

5.1. PRINCIPES GÉNÉRAUX

L'architecture logicielle se traduit par l'expression d'un ensemble de composants qui vérifient des propriétés préalablement établies et qui entretiennent des relations [Coutaz et Nigay 2001]. Une architecture logicielle trouve son importance dans des problématiques comme l'itérativité du processus de conception qui se traduit par des nécessités de "modifiabilité" ou de réutilisabilité du code, ou dans la complexité croissante des applications et du besoin de les diviser en sous-problèmes.

Dans notre cas, nous proposons une génération automatique de code produisant une application qui fonctionne. Cette génération produit une application devant s'exécuter sur tout type de plate-forme. Aussi pour minimiser le code généré et réutiliser le maximum de composants, nous devons respecter au mieux la propriété d'indépendance fonctionnelle. C'est ce qui a motivé le choix d'une architecture logicielle de référence et son respect au cours de la génération du code.

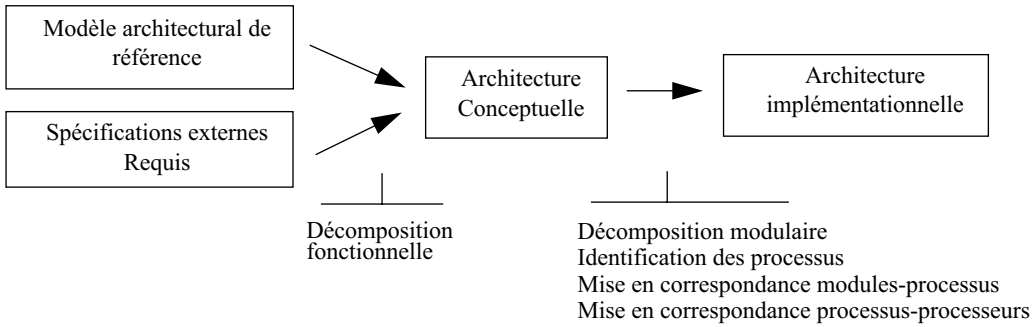
Processus : Les étapes

Le processus de génération, à partir d'un modèle d'architecture de référence jusqu'à la production du code respectant cette architecture, couvre un ensemble d'aspects illustrés par la figure suivante.

Chaque boîte montre une étape de la conception logicielle :

- Modèle architectural de référence ; C'est une décomposition fonctionnelle normalisée pour un problème connu. Nous avons retenu

Figure 18. Etapes de génération de code. Relation entre modèle architectural de référence, architecture conceptuelle et architecture implémentationnelle. Les flèches dénotent une approche descendante du processus de conception architectural. Schéma tiré de[Coutaz et Nigay 2001].



PAC-Amodeus (cf. chapitre 5) ;

- Spécifications externes et requis ; Correspondent aux spécifications de l'application. Dans notre cas, il s'agit des différents modèles de notre processus de conception (modèles des tâches, des concepts, etc.) ;
- Architecture conceptuelle ; C'est la concrétisation du modèle de référence en fonction des requis et spécifications externes. Cette structure décrit les composants du système à implémenter et leurs relations ;
- Architecture implémentationnelle ; C'est l'architecture du code généré. Elle décrit comment les composants sont réalisés — sous la forme d'une ou plusieurs classes –, comment les relations entre composants sont implémentées — appel de méthodes, événements, etc.

Le passage entre les différentes étapes du processus s'appuie sur les points suivants :

- La décomposition fonctionnelle ; Elle consiste à exprimer les besoins fonctionnels du système en des unités plus simples. Elle plaque le modèle d'architecture de référence aux spécifications externes. Le résultat de cette activité produit l'architecture conceptuelle ;
- La décomposition modulaire définit une structuration statique du système selon des règles. En relation étroite avec la décomposition modulaire, l'allocation des fonctions aux modules consiste à ranger les fonctions de l'architecture conceptuelle dans des modules. Autrement dit, il s'agit de définir la couverture fonctionnelle de chaque module : un module peut regrouper plusieurs fonctions, et une fonction peut-être répartie sur plusieurs modules ;
- L'identification des processus et l'assignation de modules aux processus consistent à décider quels modules serviront de lieu d'exécu-

tion aux processus.

Ces différentes étapes sont décrites plus en détail dans [Coutaz et Nigay 2001]. Après avoir défini la couverture du processus d'application d'une architecture logicielle, il est nécessaire de clarifier la manière dont elle peut-être mise en œuvre dans le cadre de la génération de code.

Processus : réalisation

Une architecture de référence une fois choisie, il convient de déterminer les méthodes qui permettront le passage au modèle implémentationnel. Ce cheminement ouvre de nombreux problèmes comme le respect du modèle de référence ou du modèle conceptuel. Par exemple, l'application des règles peut aboutir à un système qui ne respecte plus les spécifications initiales. L'évolution du projet lors des différentes intérations peut aboutir à une implémentation qui diverge des spécifications initiales. Les règles peuvent être ambiguës, laissant libre cours à de mauvaises interprétations.

[Anderson et al. 2000] identifient trois approches :

- l'application d'heuristiques décrites dans un guide de conception. C'est ce que propose [Nigay 1994] avec des règles s'appliquant au modèle de référence PAC-Amodeus ;
- l'utilisation d'outils s'appuyant sur un modèle architectural de référence. C'est ce que font les MB-IDE ;
- l'utilisation d'outils paramétrés ou qui permettent les retouches manuelles.

Dans ARTStudio, nous visons une génération automatique. Nous sommes donc partis vers une approche de type MB-IDE et avons appliqué des heuristiques de génération. L'outil, qui n'est pas paramétrable à ce niveau, n'offre pas la possibilité au développeur de modifier le code généré.

5.2. PRINCIPES DÉTAILLÉS

ARTStudio compile l'IUC, qui est indépendante d'un langage de programmation, en du code source. Nous présentons ici la génération de l'architecture logicielle puis la production du code exécutable.

Architecture logicielle

Au chapitre 5, nous avons présenté les deux étapes de production d'une architecture : la production de l'architecture conceptuelle, et la production de l'architecture implémentationnelle. Nous allons revenir maintenant sur ces deux étapes et vérifier la correspondance de nos heuristiques avec le modèle de référence PAC-Amodeus. ARTStudio génère des IUF en Java et en JESS. Pour plus de simplicité, nous présentons la génération sous la forme de règles en français.

Architecture conceptuelle :

Table2. Règles de production de l'architecture conceptuelle.

	Règles (AC)
R1	Un agent est créé pour la racine.
R2	Tout concept à représenter est modélisé par un agent.
R3	Tout système de navigation est modélisé par un agent. Cet agent de navigation s'occupe de l'activation de ses fils.

Table3. Corollaire aux règles de production de l'architecture conceptuelle.

	Corollaire
C1	Tout espace de travail est modélisé par un agent (implication de R1et R3)

Architecture implémentationnelle :

Table4. Règles de production de l'architecture implémentationnelle.

	Règles (AI)
R1	Tout agent est concrétisé par une classe vérifiant une interface spécifique.
R2	Tout concept est une variable active. Il hérite donc de TObject qui implémente la sémantique du Sujet (cf. "Principe de la notification" page 125).
R3	Tout agent abonné à un concept doit vérifier l'interface de notification et donc implémenter la méthode de notification. Il doit mettre à jour les informations pertinentes lors d'une notification. Il hérite donc de ActionPerformed qui est une interface de type Observateur (cf. "Principe de la notification" page 125).
R4	Tout agent représentant un concept doit être abonné en écriture sur ce concept. Il utilise la méthode "registry" de TObject.
R5	La communication du CD -> ANF -> NF se fait par appel de méthode.
R6	La communication du NF-> ANF -> CD se fait par notification grâce à la méthode "notifyMethode". L'objet du NF dont l'état a été modifié, est passé en paramètre de cette méthode.
R7	Le dialogue est construit sur des variables qui définissent son état.
R8	La communication entre agent (le dialogue) se fait par la modification des variables d'état du dialogue et la réception de notifications de changement d'état du dialogue.
R9	Un agent devant communiquer avec d'autres agents doit avoir une référence sur les variables de dialogue qui l'intéressent et éventuellement s'abonner à celles-ci.
R10	Une variable d'état du dialogue peut-être liée à une ou plusieurs autre(s) variable(s) d'état.
R11	Dans certain cas bien précis, le dialogue est tout de même effectué par appel de méthode ; ex : activation d'agent par un agent de navigation, création d'un agent.

Vérification des règles :

Table5. Confrontation des règles heuristiques de mise en œuvre de PAC-Amodeus proposées par Nigay [Nigay 1994] avec les règles dans ArtStudio

Règle	Descriptif rapide	Analyse
1	Une fenêtre qui sert de support à un espace de travail est modélisée par un agent.	**OUI** Corollaire AC 1.
2	Les vues multiples d'un même concept sont gérées par un agent "vue multiple".	**NA** Règles AI 2-3-4 répondent à ce problème sans avoir à créer un agent vue multiple.

Table5. Confrontation des règles heuristiques de mise en œuvre de PAC-Amodeus proposées par Nigay [Nigay 1994] avec les règles dans ArtStudio

Règle	Descriptif rapide	Analyse
3-4	Une palette de classes de concepts est modélisée par un agent	**OUI** Agent de réalisation de la tâche sélection (issue des spécifications externes : arbre de tâche)
4	Une barre de menus est modélisée par un agent	**OUI** Une barre de menu est un système de navigation donc OK d'après règle AC 3. Mais nous allons plus loin puisque tout système de navigation est modélisé par un agent.
5	Une zone d'édition est modélisée par un agent.	**OUI/NA** La problématique de la manipulation directe n'est pas traitée. Sinon la règle AC 2 assure cet aspect
6	Un concept complexe d'une zone d'édition est modélisé par un agent.	**OUI** Règle AC 2. Notre règle est plus forte : tout concept est représenté par un agent
7	Si, depuis une fenêtre "espace de travail" contenant un concept de classe donnée, il est possible d'ouvrir un espace de travail sur un autre exemplaire de la même classe, ces deux espaces sont modélisés comme des agents fils d'un agent "père" commun.	**NA** La génération du dialogue est dirigée par la structure des tâches.
8	Si, depuis une fenêtre d'édition qui restitue, de manière synthétique, un ou plusieurs concepts composés, il est possible d'ouvrir une nouvelle fenêtre représentant plus en détail le contenu d'un de ces concepts composés, l'agent qui modélise la nouvelle fenêtre est fils de l'agent source.	**NA** Comme pour R7, cela dépendra de l'arbre de tâches.
9	Si la spécification d'une commande (ou tâche élémentaire) implique des actions distribuées sur plusieurs agents, ceux-ci doivent être placés sous le contrôle d'un agent "ciment-syntaxique" qui synthétise les actions réparties en une unité d'information de plus haut niveau d'abstraction.	**OUI** Règle implicite car directement issue de la spécification des tâches. C'est l'un des sens d'une tâche abstraite
10	Un agent et un agent fils unique peuvent être regroupés en un seul agent	**NA** Ce genre d'optimisation n'est pas géré
11	Un agent dont la fonction est réalisée par un objet de présentation ou d'interaction d'une boîte à outils peut-être éliminé de l'arborescence et cet objet de présentation prend part à la Présentation de l'agent père de l'agent éliminé.	**NA** Nous n'utilisons aucun agent d'une boîte à outils. Ils sont tous spécifiques car ils implémentent les techniques de notification vues précédemment.

Production du code Un agent est réifié en une classe qui est une spécialisation de fenêtre ou de canevas.

- En Java, une fenêtre est un « Interactors.XframeAndValidation » et un canevas est un « Interactors.Xpanel ».
- En Waba, une fenêtre est un « WInteractors.XframeAndValidation »

et un canevas, un « WInteractors.Xpanel ».

Une XframeAndValidation est une fenêtre avec boutons de validation (Okay/Annuler).

ARTStudio utilise par défaut ce principe de validation. Aussi tout concept manipulé dans une fenêtre est une copie du concept dans le NF. Au moment de la validation, le concept du NF est mis à jour avec la copie. Si l'utilisateur annule, alors le NF n'est pas modifié.

Pour chaque classe générée :

Deux constructeurs sont créés :

1. le premier avec comme seul argument, l'instance de l'agent père ;
2. le second avec comme arguments toutes les instances des concepts externes dont la portée touche cet agent (cf. spécification du modèle des tâches/Concepts).

Des méthodes sont définies :

1. les méthodes définissant les instances des concepts externes ;
2. les méthodes de communication «notifyMethode» et «actionPerformed» ;
3. les prologue et épilogue qui sont appelées à la construction de l'objet et à sa destruction.

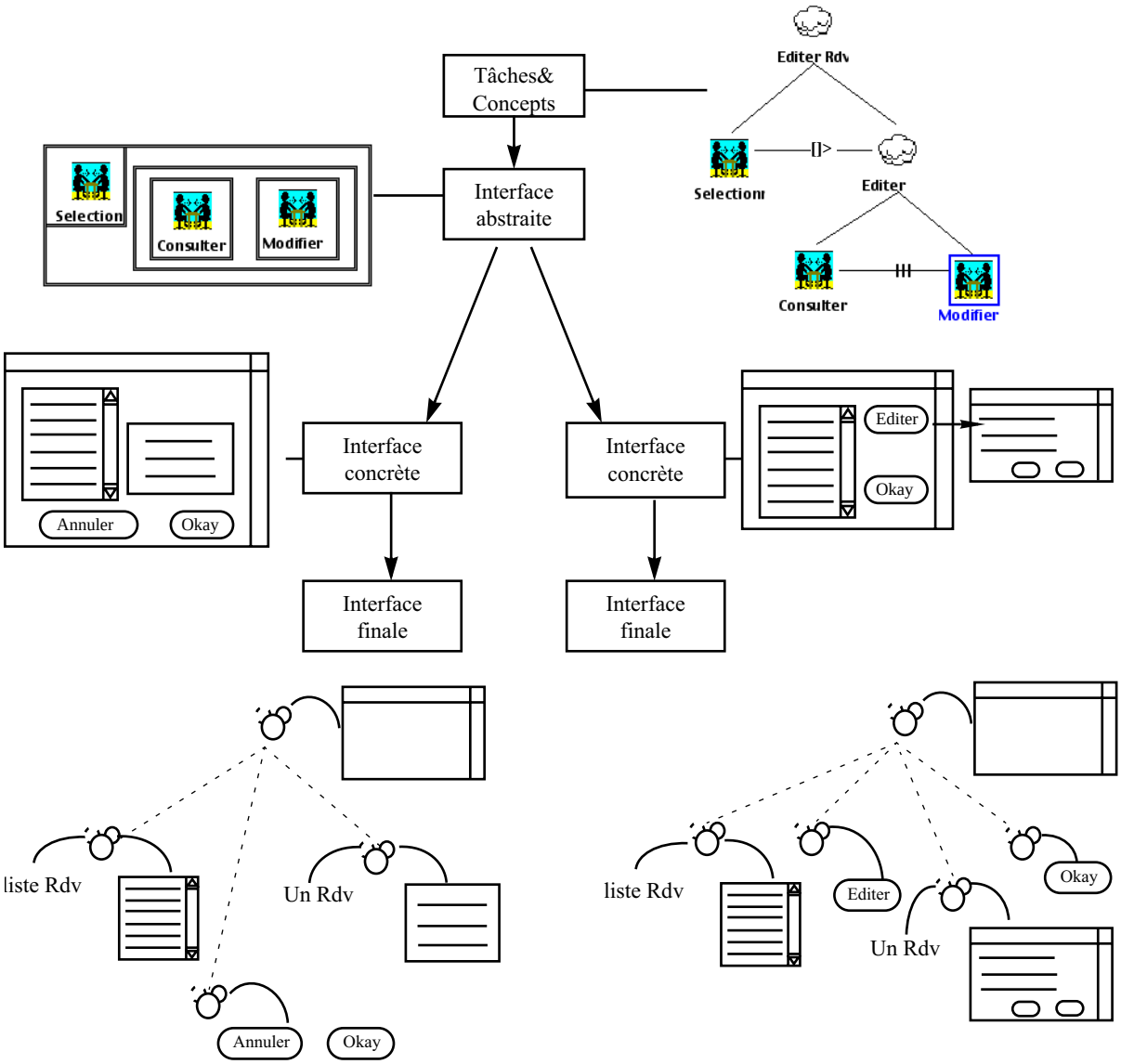
Lors de la génération de l'interface finale, sont produits autant de fichiers que de plates-formes cibles.

6. Synthèse

ARTStudio propose une réalisation possible de notre cadre de référence pour la génération d'interface multicible. Il vise une production d'interface Java/Swing (pour station de travail) et Waba (PalmPilot ou PocketPC). Le processus, dont un exemple est donné figure 19, s'appuie sur une définition des tâches et des concepts pour produire une IHM indépendante de la plateforme cible : c'est l'Interface Utilisateur Abstraite. Cette IUA est concrétisée pour chaque cible en l'Interface Utilisateur Concrète. Il s'agit d'un prototype d'interface graphique. Ce prototype est compilé, c'est-à-dire transformé en code source décrivant l'interface et les liens avec le NF : c'est l'Interface Utilisateur Finale.

L'expérience montre la nécessité d'un résolveur de contraintes puissant. Le résolveur actuel est insuffisant : trop sous-contraint (il ne gère pas

Figure 19. Processus en fermeture éclaire pour la conception et génération d'IHM multicible.



tous les types de contraintes – en particulier les contraintes géométriques et ergonomiques), il calcule des milliers d'interfaces et rend, par conséquent, nécessaire l'adoption de critères d'arrêt arbitraires tels que "s'arrêter à la première solution".

De manière générale, les règles de génération utilisées dans ARTStudio sont très simples. En fait nous avons axé notre travail sur la mise en œuvre d'un outil de conception multiplate-forme sans trop investir sur la qualité des règles de génération et du moteur d'inférence.

Le prototype MMS avait pour but de tester des règles d'adaptation plus poussées comme le choix entre différents interacteurs, différentes présentations, ou des règles ergonomiques comme la cohérence, pour

ensuite les intégrer dans ARTStudio. Aujourd'hui cette intégration n'a pas été faite.

Si nous avons bien avancé sur le processus de réification, nous avons peu investi sur la traduction. MMS inclut la traduction en changeant les interacteurs de présentation, mais ARTStudio n'a pas bénéficié de ces algorithmes. La traduction aux autres niveaux de réification se limite à une copie sans traitement.

Nous reviendrons sur les limites de notre production technique au chapitre suivant qui conclut ce mémoire.

Chapitre 7 *Conclusion*

Avec le concept de plasticité, nous ouvrons explicitement un nouveau terrain de recherches en accord avec les besoins émergents. Ces besoins, que résume le slogan “accès à l’information partout, à tout moment, par tous”, sont déjà réalité : le gestionnaire d’énergie d’EDF qui a servi d’application à notre travail, notre projet de post-doctorat au NII, les applications WAP et son équivalent japonais I-Mode, en sont les témoins.

Ce mémoire consigne une première contribution au développement d’IHM multicibles, aux IHM plastiques en particulier. Nous en rappelons ici les points essentiels avec leurs forces et faiblesses qui appellent à leur tour, des prolongements à court et moyen terme.

1. Résumé de notre contribution

Nous avons centré notre réflexion sur les outils de production d’IHM multicible, laissant au second plan, sans toutefois les ignorer, les mécanismes nécessaires à l’adaptation dynamique. Ce choix trouve sa principale raison dans notre volonté de répondre aux difficultés de développement et de maintien des versions d’une même IHM pour différentes cibles. Par rapport à cet objectif, notre contribution comprend deux volets complémentaires : un volet conceptuel structurant, et un volet technique démontrant la faisabilité de nos propositions conceptuelles.

Sur le plan conceptuel, nous proposons un cadre de référence servant le processus de conception d’IHM multicible. Ce cadre, orienté génie logiciel, s’appuie sur cinq principes :

- les principes de description, de factorisation et de décoration qui favorisent la capitalisation et la réutilisation des connaissances, de même que la minimisation des efforts du maintien de la cohérence

intercible;

- les principes de réification et de traduction qui correspondent aux fondements de la génération et de l'adaptation d'IHM multicible.

Sur le plan technique, nous proposons ARTStudio, un outil de génération d'IHM multiplate-forme, et deux prototypes, MMS et CUIDesigner :

- ARTStudio démontre la mise en application de notre cadre conceptuel de référence, même s'il ne couvre pour l'instant qu'une classe particulière d'IHM multicible. Il permet la génération d'IHM exécutables Java/Swing (pour Macintosh ou PC par exemple) et Waba (pour PalmPilot ou PocketPC).
- MMS et CUIDesigner ont permis de tester et de valider, de manière complémentaire à ARTStudio, d'autres aspects de l'adaptation. MMS propose une adaptation dynamique par traduction et utilise des règles d'inférence plus complexes que celles implémentées dans ARTStudio. CUIDesigner peut se voir comme une extension d'ARTStudio pour la gestion d'un modèle de présentation d'interface à partir de contraintes de placement.

2. Originalité et points forts

L'originalité et les points forts de notre contribution se situent au niveau de l'approche conceptuelle et des réalisations techniques.

2.1. VOLET CONCEPTS

L'originalité du cadre de référence pour le développement d'IHM multicible tient aux principes sur lesquels il s'articule. Ces principes sont tous motivés par des besoins récurrents en conception.

Le principe de factorisation traduit la capitalisation et le contrôle des différences entre versions d'un même modèle. Si cette idée de contrôle de version est aujourd'hui un problème largement traité en génie logiciel, en conception d'IHM et en particulier en conception multicible, il n'a été que très peu appliqué. Un premier point fort de notre travail est d'appliquer ce principe à toutes les étapes du processus de développement.

Le principe de décoration est une seconde originalité. Dans notre objectif de description et de contrôle des différences entre cibles, ce principe facilite l'expression des cas particuliers, évitant au concepteur de produire des descriptions spécifiques pour chaque exception. Alors que le principe de factorisation propose une approche globale à l'expression des différences, la décoration permet l'ajustement par "touches" locales.

Le troisième point fort de notre cadre tient à la coopération des principes de réification et de traduction tout au long du processus de génération/adaptation. Si ces deux principes existent séparément et de manière implicite dans des outils, nous les avons clairement définis et combinés pour créer une nouvelle manière d'appréhender le développement multicible. Maintenant, le développement d'IHM multicible doit se voir comme une génération verticale, une génération horizontale, et des relations entre ces différentes étapes de génération.

2.2. VOLET RÉALISATION TECHNIQUE

Le deuxième point fort de cette thèse est la mise en œuvre d'un outil de génération d'interface multicible, et du prototype MMS testant une adaptation dynamique des interfaces.

ARTStudio

Bien qu'il réalise un sous-ensemble de notre cadre de référence, ARTStudio couvre tout le processus de génération, depuis la description des tâches et concepts du domaine, à la production du code exécutable d'une application fonctionnelle (c'est-à-dire sans simulation car reliée effectivement à un noyau fonctionnel). L'outil complet, avec les différentes bibliothèques, représente 50000 lignes de code Java.

Prototype MMS

Le deuxième point fort de cette thèse est le développement d'un prototype testant plus finement des règles et moteurs d'inférences complexes dans le but d'améliorer la génération dans ARTStudio.

MMS s'appuie sur des règles et une architecture logicielle, pour fournir une adaptation dynamique de l'interface. Le système s'adapte en fonction des interacteurs disponibles, de la surface d'affichage et vérifie la propriété de cohérence de l'interface.

3. Limites

Les limites de notre thèse tiennent principalement à la mise en œuvre. De par l'objectif ambitieux que nous visions avec ARTStudio, nous avons mis l'accent sur un générateur qui couvre toutes les étapes de réification de notre cadre conceptuel, pour revenir ensuite sur des difficultés identifiées au cours de nos travaux. En particulier :

- ARTStudio ne couvre pas tous les principes du cadre conceptuel. Il ne gère pas le principe de décoration et n'inclut pas d'outils et mécanismes de traduction ;
- les règles de génération (d'adaptation) sont simples, mais celles testées avec MMS pourraient être intégrées ;
- ARTStudio ne gère qu'une génération d'interface graphique et suppose présents les dispositifs d'interaction usuels comme le clavier,

la souris ou le stylet, l'écran.

- ARTStudio n'inclut pas le modèle d'environnement. Il ne génère donc pas d'IHM capable d'adaptation à l'environnement.

Ces lacunes justifient nos perspectives à court et à long terme.

4. Perspectives

4.1. A COURT TERME

Nous disposons d'un cadre de conception dont la portée n'a pas été complètement explorée (et notamment la traduction). Aussi nos perspectives à court terme sont plutôt d'ordre implémentationnelle pour compléter la validation de notre cadre de référence et proposer une génération plus "intelligente".

Règles pour l'inférence

L'un des principaux défauts d'ARTStudio est d'utiliser des règles d'inférence très simples pour la génération des descriptions transitoires. Un premier travail est d'étendre le corpus pour améliorer cette génération. Des systèmes comme Trident ont déjà montré la faisabilité d'utilisation de règles ergonomiques pour la génération. Mais il convient d'envisager d'autres règles portant par exemple sur l'enchaînement des tâches.

Moteur d'inférence

Nous avons constaté la simplicité de nos règles de génération. Etendre le corpus de règles passe également par l'intégration de moteurs d'inférence plus performants, gérant différentes classes de contraintes, etc. Nous avons par exemple vu la limite d'un moteur comme Cassowary du fait de l'incomplétude des solutions proposées. Aussi une extension d'ARTStudio serait de coupler JESS à Cassowary, ou encore d'intégrer un système d'inférence comme Prolog IV.

4.2. A LONG TERME

Notre vision à plus long terme concerne essentiellement l'extension de la couverture des cibles visées (multimodalité et environnement), l'adaptation dynamique et le développement d'outils de prototypage et de rétroconception.

Multimodalité

Aujourd'hui, ARTStudio est limité à une génération d'interface graphique et suppose la présence des dispositifs d'interaction usuels (clavier, écran, souris ou eq.). Il convient donc d'envisager une génération d'interface multimodale, couplée à la découverte dynamique des res-

sources d'interaction et des conditions environnementales. Ainsi ARTStudio devrait pouvoir générer une interface :

- WML pour téléphone WAP ;
- sonore pour téléphone "normal" ;
- reconnaissance de la parole pour un système mobile dans un environnement empêchant l'utilisation des mains ;
- etc.

Cette extension implique, entre autres, une révision de nos modèles de plate-forme et d'interacteurs, prolongée de nouvelles règles de génération dans le respect de l'utilisabilité (N'oublions pas que la plasticité implique une adaptation qui doit respecter les propriétés d'utilisabilité fixées à l'expression des requis).

Environnement

Nous n'avons pas traité la dimension "environnement" de la plasticité. Cet aspect est traité par ailleurs dans l'équipe avec la notion de contexteur [Rey 2001]. Il conviendra de tester les modèles de ARTStudio au regard de cette extension.

Adaptation dynamique

Avec MMS, nous avons fait un premier pas vers l'adaptation dynamique. A court terme nous aimerions continuer dans cette voie pour proposer une architecture plus poussée dans la gestion de l'adaptation dynamique. A plus long terme, ce travail devrait se projeter dans ARTStudio, au niveau de la production de l'interface utilisateur finale. Ainsi ARTStudio permettrait de faire de l'adaptation précalculée et/ou dynamique. Mais l'adaptation dynamique nécessite des mécanismes de capture de l'état du système sous-jacent, environnemental ou utilisateur. Il conviendrait de travailler sur la proposition de mécanismes standard, à la manière de JINI, qui informent l'application de l'évolution du contexte d'interaction. Ces mécanismes d'introspection, devraient être intégrés au niveau du système d'exploitation.

Rétroconception et prototypage

Parallèlement à ces objectifs en relation directe avec notre thèse, nous aimerions travailler sur un outil permettant le prototypage rapide et la rétroconception. Nous pensons qu'il s'agit d'un réel besoin et que l'une des raisons de l'échec de l'approche MB-IDE est le carcan imposé par sa rigidité. Or, des outils de prototypage rapide associés à de la rétroconception faciliteraient l'utilisation d'un outil comme ARTStudio. L'idée générale, qui a été initiée dans ARTStudio à travers les éditeurs graphiques de descriptions, est de cacher le plus possible tous les aspects de la formalisation nécessaires à une génération automatique d'interface, et d'"assouplir" l'utilisation d'outils de génération en proposant le prototypage.

Les outils de rétroconception, à la manière de Vanderdonckt et al. [Vanderdonckt et al. 2001] visent un cheminement vertical de bas en

haut du processus de réification. Il serait intéressant d'étendre notre cadre conceptuel pour tenir compte de ce requis.

5. Synthèse

Au bilan, ces travaux de recherche doctorale ont contribué à définir les fondements conceptuels pour le développement d'IHM multicible, d'IHM plastique en particulier. La mise en œuvre technique d'outils qui répondent aux requis de développement IHM plastiques a révélé de nombreuses exigences que nous ne pouvions remplir dans le cadre de cette thèse. Ces exigences ouvrent des pistes d'approfondissement faisant appel à des connaissances multiples non seulement en Interaction Homme-Machine mais en système d'exploitation : interaction multimodale, mécanismes de détection de changement de contexte d'interaction, découverte de ressources interactionnelles, IHM transitoires d'adaptation, pour ne citer que quelques nouvelles niches de recherche.

Annexe 1 : Propriétés

La conception d'interfaces utilisateur et leur évaluation passent par la définition et la vérification de propriétés. Elles sont de deux types : les propriétés externes et internes à l'application [Gram et Cockton 1997] [Dix et al. 1998]. Parmi ces propriétés, se trouvent celles caractérisant un système interactif pouvant faire de l'adaptation.

Dans cette annexe, nous rappelons (définissons) et classons ces propriétés dont les définitions peuvent se recouvrir. Nous verrons tout d'abord les propriétés externes, puis les propriétés internes.

1. Propriétés externes

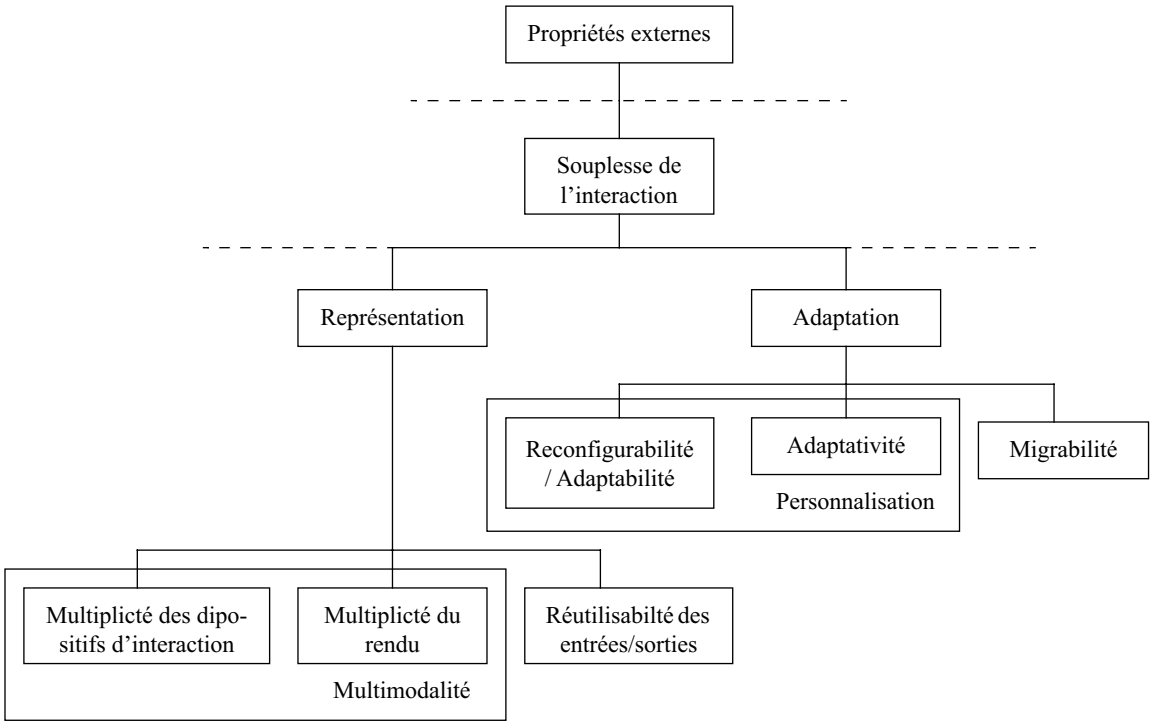
L'utilisabilité d'un système interactif est lié à la qualité du dialogue. Cette qualité peut-être mesurée par des propriétés du dialogue [Gram et Cockton 1997]. Il s'agit d'un ensemble de propriétés centrées utilisateurs : ce sont les propriétés externes.

Parmi elles se trouvent celles caractérisant la souplesse de l'interaction. Nous trouvons dans cette classe, les propriétés caractérisant l'adaptation d'un système interactif. La figure 1 montre une classification hiérarchique de ces propriétés.

Définition. La **souplesse** (*flexibility*) caractérise ce qui s'adapte aisément aux circonstances. [Gram et Cockton 1997] définissent cette propriété comme la capacité du système à offrir plusieurs chemins entre l'utilisateur et le système pour l'échange d'information lors de l'interaction. Elle exprime l'éventail des choix (pour l'utilisateur et le système) [Coutaz et Nigay 2001].

Définition. L'**adaptativité** (*adaptativity*) est la propriété caractérisant une interface utilisateur adaptative.

Figure 1. Classification des propriétés externes liées à l'adaptation des interfaces.



Définition. Une interface utilisateur **adaptative** (*adaptive user interface*) est une interface capable de changer son comportement automatiquement pour correspondre à un individu ou groupe d'individus [Browne et al. 1990b].

Définition. La **reconfigurabilité** est la capacité du système à supporter un ajustement, une modification de l'interaction en entrée ou en sortie, fait par l'utilisateur [Gram et Cockton 1997].

Définition. L'**adaptabilité** (*adaptability*) est la propriété caractérisant une interface utilisateur adaptable.

Définition. Une interface utilisateur est dite **adaptable** (qui peut être adaptée) lorsqu'elle peut-être modifiée par l'utilisateur selon ses besoins [Opperman et Simm 1994].

Alors que l'adaptabilité et l'adaptativité se différencient habituellement par l'acteur de l'adaptation, [Stephanidis et al. 2001] adoptent des définitions différentes, en les caractérisant selon un point de vue acquisition de connaissances nécessaires à l'adaptation.

Définition. Pour [Stephanidis et al. 2001] l'**adaptabilité** caractérise une interface capable de s'adapter en utilisant les connaissances déterminées avant l'exécution. Cette adaptation est automatique.

Définition. Pour [Stephanidis et al. 2001] l'**adaptativité** caractérise une interface capable de capturer les informations pour son adaptation, en cours d'exécution. Cette adaptation est automatique.

Définition. La **migrabilité** est la capacité d'un système à supporter le transfert de tâches utilisateur vers le système et inversement. Ce transfert peut-être initié par le système ou par l'utilisateur [Gram et Cockton 1997].

Définition. La **personnalisation** (*Cutomizability* [Dix et al. 1998]) est l'action de donner un caractère personnel à l'interface. La personnalisation peut-être faite soit par l'utilisateur, soit par le système. Elle suppose l'adaptabilité et l'adaptativité.

Définition. La **multiplicité des dispositifs d'interaction** est la capacité du système à proposer différents dispositifs d'interaction d'entrée ou de sortie. Par exemple en entrée nous aurons des dispositifs comme le microphone, le clavier, la souris, la caméra vidéo, etc. et en sortie nous aurons l'écran, les hauts parleur, etc. Les systèmes haptiques sont à la fois d'entrée et de sortie.

Définition. La **multiplicité du rendu** (représentation multiple) est la capacité du système à fournir plusieurs représentations pour un même concept.

Définition. Un système est dit **multimédia** lorsqu'il utilise ou permet d'utiliser plusieurs médias de communication (texte, son, images fixes ou animées).

Définition. La **multimodalité** peut être définie comme l'utilisation de plusieurs modalités séquentiellement et/ou parallèlement. Une modalité désigne un type de qualité sensorielle. Cette définition reste vague car elle varie selon les domaines (cf. "Boîte à outils multimodale") page 32.

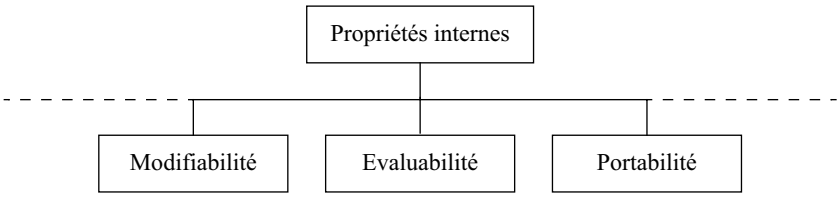
Définition. La **réutilisabilité** des données d'entrée et de sortie est la capacité de l'interface, de réutiliser des données automatiquement : les sorties du système peuvent être utilisées comme des données d'entrée (par exemple le couper-coller), ou les entrées de l'utilisateur peuvent être réutilisées par le système en sortie (par exemple les valeurs par défaut).

2. Propriétés internes

Alors que les propriétés externes définissent l'aspect visible par l'utilisateur de l'interface homme-machine de son application, les propriétés

internes traduisent des propriétés du code de l'application. La mise en oeuvre de systèmes capables de s'adapter, passe par le respect de certaines de ces propriétés internes au système. La figure 2 présente ces trois propriétés.

Figure 2. Classification des propriétés internes liées à l'adaptation des interfaces.



Définition. La **modifiabilité** traduit la facilité d'un système interactif à être modifié en vue d'une évolution. C'est un facteur important pour augmenter le cycle de vie d'un logiciel.

Définition. L'**évaluabilité** traduit la facilité du logiciel à être évalué. L'évaluation est une vérification de propriétés (ex : maintenabilité du code, visibilité, etc.).

Définition. La **portabilité** (*portability*) est l'aptitude d'un logiciel ou d'un matériel à être utilisé sur des systèmes informatiques différents. Comme l'a exprimé [Thevenin 1998] cette propriété n'est pas suffisante pour faire de l'adaptation.

Annexe 2 : ARTStudio

Dans cette annexe, nous présentons ARTStudio, un assistant logiciel pour la production d'interfaces plastiques. ARTStudio se décline en deux outils :

- PlasticityApp, un générateur automatique, utilisable en ligne de commande. PlasticityApp n'offre aucun environnement d'édition des modèles. Les modèles peuvent être édités manuellement, hors PlasticityApp, via un éditeur de texte, par exemple.
- ARTStudioApp, un environnement de gestion, d'édition et de génération des modèles impliqués dans la production d'interfaces plastiques. ARTStudioApp permet de spécifier les modèles des tâches, concepts et liens entre tâches et concepts (spécification orientée tâches), de visualiser le modèle de l'IHM abstraite et d'éditer le modèle de l'IHM concrète.

Nous allons présenter :

1. l'utilisation de ces deux outils ;
2. la réalisation de l'outils de génération ;
3. des exemples d'utilisation du code pour créer de nouveaux modèles.

1. Les outils

PlasticityApp et ARTStudioApp adoptent une même logique de fonctionnement. Ils diffèrent par leurs logiques d'utilisation. Nous en décrivons ici la logique de fonctionnement puis les logiques d'utilisation.

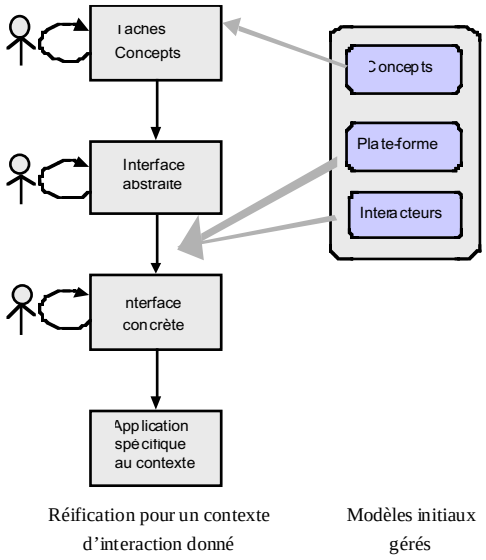
1.1. LOGIQUE DE FONCTIONNEMENT

Après avoir présenté le processus de développement implémenté dans ARTStudio, nous décrivons ici les modèles impliqués ainsi que leurs heuristiques de génération.

Processus de développement. Le processus de génération implémenté dans ARTStudio est une instance du processus de référence décrit en partie II (figure 1) :

- Le modèle des tâches n'est pas explicite. C'est un modèle Tâche-Concepts (spécification orientée tâches) référençant déjà les concepts manipulés dans les tâches ;
- Les modèles de l'environnement et d'évolution ne sont pas intégrés ;
- Les modèles de la plate-forme et des interacteurs sont référencés à partir de l'interface abstraite pour la génération de l'interface concrète.

Figure 1. Processus de développement implémenté dans ARTStudio.



D'un point de vue multiplates-formes, deux mécanismes sont implémentés (figure 2)

- Une réification avec point de divergence entre les IHMs Abstraites et concrète. C'est le mécanisme mis en œuvre pour la production d'interfaces graphiques Mac/PC et Pilot, dont les interacteurs diffèrent (d'où l'ouverture de la fermeture éclair) ;
- Une traduction par simple copie. C'est le mécanisme mis en œuvre pour créer un brin de réification propre aux plates-formes WAP.

PlasticityApp et ARTStudioApp respectent cette instance du processus (figure 1 et figure 2). ARTStudioApp l'organise autour de la notion de projet.

Notion de projet Un projet est une structure qui consigne les fichiers nécessaires à la génération. La figure 3 en montre un exemple. Il s'agit d'un extrait du fichier «projets/empty/empty.x ».

Figure 2. Processus de développement. Il est implémenté dans ARTStudio pour la production d'interfaces multi-plates-formes.

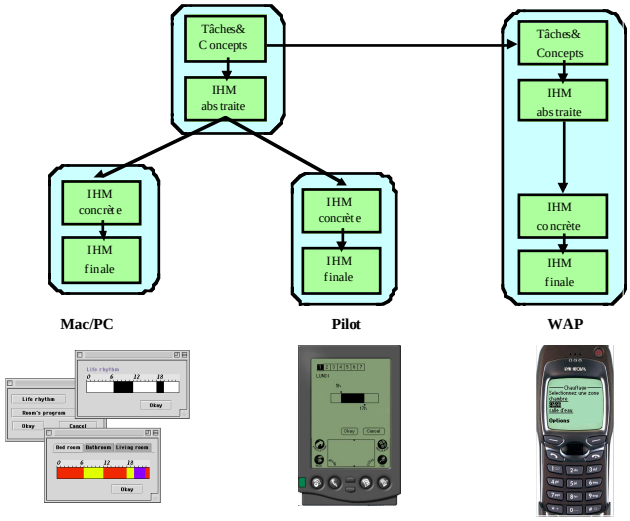


Figure 3. Un exemple de projet.

```
<projectModels>
  <TaskModel>
    <name>empty</name>
  </TaskModel>
  <ConceptModel>
    <name></name>
  </ConceptModel>
  <AbstractUIModel>
```


Un projet complet comporte neuf fichiers :

- Le fichier du projet « name.x » ;
- Le fichier de l'arbre de tâches « name.xmx » ;
- Le fichier des concepts « name.xmi » ;
- Le fichier de l'IHM abstraite « name.xaui » ;
- Le fichier de l'IHM concrète « name.xmu » ;
- Le fichier de description de la plate-forme « name.xpf » ;
- Le fichier de description des interacteurs « name.xim » ;
- Ainsi que deux fichiers « name.xmp » et « name.xauip » respectivement nécessaires à ARTStudio pour la présentation graphique du modèle de tâches et de l'IHM abstraite.

A sa création, un projet doit pointer vers un modèle des tâches comportant au moins une tâche. Les autres modèles sont rajoutés au fil de la réification.

Chaque projet est sauvé en un fichier texte au format XML.

Modèles et heuristiques

Tous les modèles sont des fichiers texte au format XML. Les DTD sont accessibles par Internet sur le site suivant :

http://iihm.imag.fr/thevenin/ProjetX/dtd/v1/. Elles sont également disponible dans le répertoire /dtd/.

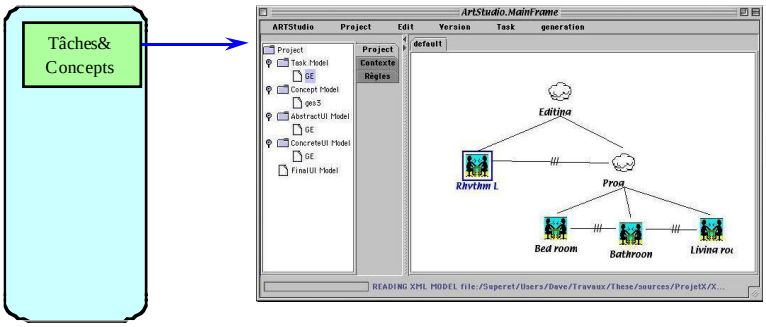
Concepts. Le modèle des concepts est une spécification UML au format XML. La spécification doit être faite avec ArgoUML [ArgoUML]. ARTStudio génère plusieurs fichiers dont un fichier « name.xmi ». C'est ce fichier qui contient la spécification des concepts.

Tâches et concepts. Le modèle Tâches-Concepts s'appuie sur la notation ConcurrTaskTree. C'est un modèle des tâches augmenté des concepts manipulés dans les tâches. Les contraintes (conventions) sont les suivantes :

- Un seul concept manipulé par tâche ;
- Tout concept est référencé par son instance (et non sa classe);
- Toute tâche est identifiée par son nom (les tâches doivent donc avoir des noms différents, servant d'identifiant unique).

Le modèle Tâches-Concepts est sauvé dans le fichier «name.xmx».

Figure 4. Modèle Tâches-Concepts. Cette figure présente l'interface d'ARTStudio pour la spécificaiton de stâches et des liens entre tâches et concepts.



Interface Utilisateur Abstraite. Aujourd'hui ARTStudio ne permet pas de modifier l'IUA générée. La figure 5 montre l'interface dans ARTStudio.

Interface Utilisateur Concrète. La figure 6 montre l'interface d'édition de l'IUC.

Figure 5. Modèle de l'IHM Abstraite. Cette figure présente l'interface d'ARTStudio pour la visualisation de l'IUA.

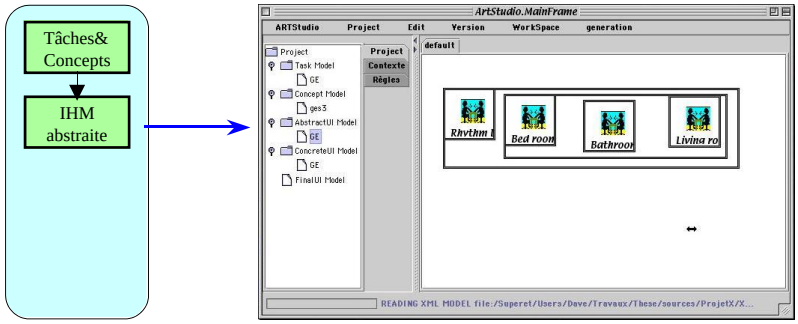
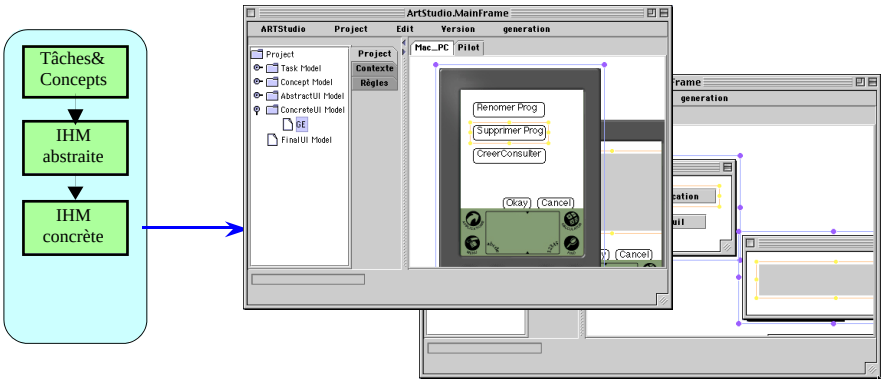


Figure 6. Interface d'édition de l'Interface Utilisateur Concrète. Cette figure présente l'interface d'ARTStudio pour la visualisation/modification de l'IUC.



Interacteurs. Seuls les interacteurs graphiques sont aujourd'hui traités. Pour chaque interacteur sont spécifiés :

- Un nom ;
- Une surface d'affichage ;
- Les concepts qu'il est capable de représenter ;
- Les tâches qu'il implémente ;
- Son type *visualisation* ou *navigation*.

La figure 7 en présente un exemple.

Figure 7. Spécification d'un interacteur.

```
<GraficInteractor>
   <name>XComboBox</name>
   <width value="null"/>
   <height value="20"/>
   <data list = "{TString,TStringFromTVector}"/>
   <elementaryTask list = "{selectionner}"/>
   <type value="visualisation"/>
 </GraficInteractor>
```

Plate-forme. Aujourd'hui seul le langage d'interaction cible et la surface d'affichage sont spécifiés. La figure 8 en présente un exemple.

Figure 8. Spécification d'une plate-forme.

```
<Container id="S.103">
   <versionName value="Pilot"/>
   <predVersionID id="S.101"/>
   <subVersionIDs listID="{}"/>
     <containerElement>
       <Language value="Waba"/>
       <Ressources>
         <screenWidth value="160"/>
         <screenHeight value="160"/>
         <screenResolution value="1"/>
    </Ressources>
  </containerElement>
 </Container>
```

1.2. LOGIQUE D'UTILISATION

Nous décrivons ici les logiques d'utilisation de PlasticityApp et ARTStudioApp.

PlasticityApp. La commande d'exécution de l'application est la suivante :

java -cp ARTStudioApp.jar:../CommonClasses/ArgoUML.jar:../CommonClasses/Jess6.jar:../CommonClasses/Concepts.jar:../CommonClasses/xml4j-2.jar:../CommonClasses/Models.jar PlasticityMain projectName.x -acf

Le projectName peut prendre comme valeurs :

- /ptittet/Users/thevenin/projets/GE/GE.x
- ou projets/GE/GE.x, etc.

Les options sont les suivantes :

-a : génération de l'IHM abstraite ;
-c : génération de l'IHM concrète;
-f : génération de l'IHM finale ;
-ac : génération des IHM abstraite et concrète ;
-cf : génération des IHM concrète et finale ;
-acf : génération des IHM abstraite, concrète et finale ;
-s : sauvegarde le résultat (attention efface alors l'ancien projet).

ARTStudioApp. La commande d'exécution de l'application est la suivante :

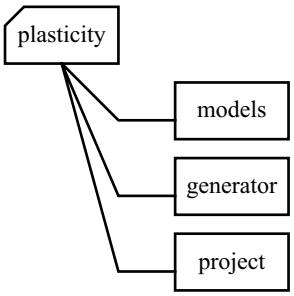
java -cp ARTStudioApp.jar:../CommonClasses/WabaAWT.jar:../CommonClasses/ArgoUML.jar:../CommonClasses/Jess6.jar:../CommonClasses/Concepts.jar:../CommonClasses/InteractorsSwing.jar:../

CommonClasses/InteractorsWaba.jar:../CommonClasses/xml4j-2.jar:../CommonClasses/InteractorsEDFSwing.jar:../CommonClasses//ConceptsEDFJDK.jar:../CommonClasses/Models.jar ARTStudioApp

ou ./runARTStudio.sh

2. Implémentation d'ARTStudio

Le package plasticity se décompose en trois sous-packages :



2.1. LE PACKAGE « MODELS »

Les modèles implémentés dans ARTStudio sont au nombre de 11 :

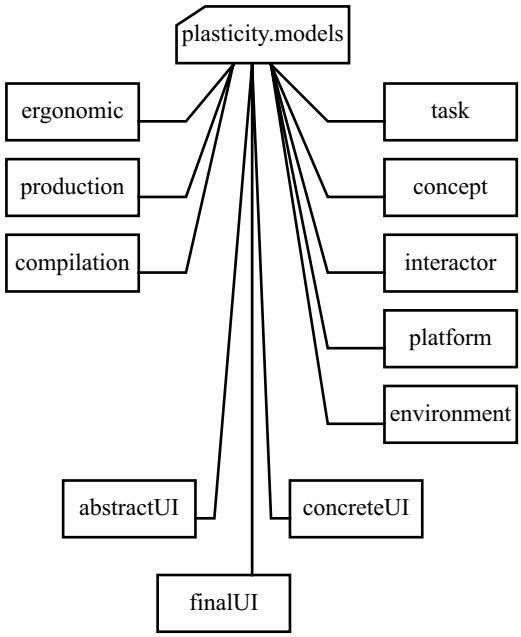
1. le modèle des tâches “plasticity.models.task.TaskModel” ;
2. le modèle des concepts “plasticity.models.task.ConceptModel” ;
3. le modèle des interacteurs “plasticity.models.task.InteractorModel” ;
4. le modèle de la plate-forme “plasticity.models.task.InteractorModel” ;
5. le modèle de l’environnement “plasticity.models.task.EnvironmentModel” ;
6. le modèle des règles de production “plasticity.models.task.ProductionModel” ;
7. le modèle des règles ergonomiques “plasticity.models.task.ErgonomicModel” ;
8. le modèle des règles de compilation “plasticity.models.task.CompilationModel” ;
9. le modèle de l’IHM abstraite “plasticity.models.task.AbstractUIModel” ;
10. le modèle de l’IHM concrète “plasticity.models.task.ConcreteUIModel” ;
11. et le modèle de l’IHM finale “plasticity.models.task.FinalUIModel”.

Un douzième modèle existe. C’est le modèle du projet. Il décrit un projet, c’est-à-dire qu’il fait le lien entre les onze modèles.

Il est à noter que les modèles 5 à 8 ne sont pas utilisés :

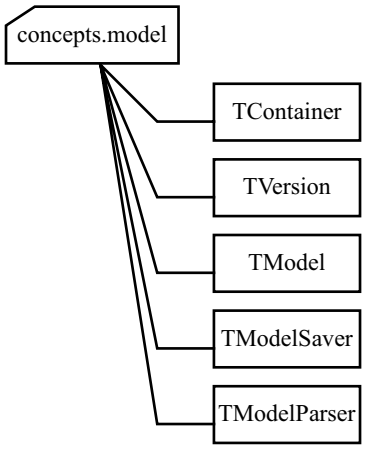
- L'environnement n'est en effet pas modélisé dans l'état actuel des travaux ;
- Les règles de production sont codées dans des fichiers à part (*.clp) qui sont utilisés par le système expert JESS. Actuellement nous avons deux fichiers :
  – jess/AbstractUIGen.clp
  – jess/ConcreteUIGen.clp
  Ils contiennent respectivement les règles permettant la production des IHM abstraite et concrète ;
- Aucune gestion de règles ergonomiques n'est pour le moment intégrée dans ARTStudio ;
- Les règles de compilation sont codées en Java directement dans notre outil.

Tous les modèles ARTStudio :

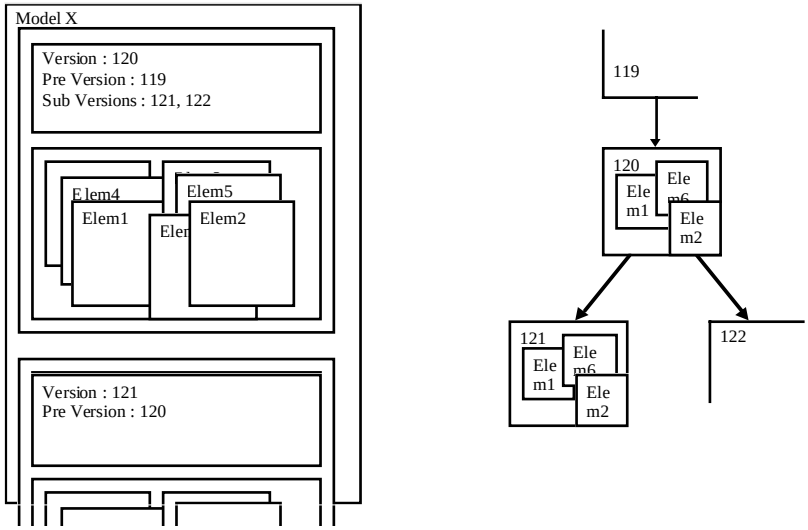


Un modèle Tout modèle ARTStudio est une spécialisation du modèle décrit dans le package "concept.model".

Ce modèle est composé de cinq classes : TModel, TModelParser, TModelSaver, TContainer, TVersion :



Un modèle est intrinsèquement multiversions. Ainsi il se décompose en sous-modèles appelés « TContainer » . Chaque TContainer pointe sur une structure TVersion et sur une table de hachage qui contient les éléments décrivant le modèle.



Chaque modèle peut implémenter une classe de sauvegarde et/ou de lecture. Il hérite alors des classes TModelSaver et TModelParser, qui respectivement sauvegarde et lit un modèle dans fichier au format TXT. Ce fichier est sauvé sous la forme d'un langage de type XML.

Nous montrons maintenant comment utiliser les classes TModel, TModelSaver et TModelParser pour implémenter un modèle pour la plasticité.

TmodelSaver TModelSaver est une classe abstraite, implémentant la sauvegarde d'un modèle ARTStudio (TModel). Le fichier est sauvé au format texte, en utilisant une structure de type XML.

Implémenter sa propre classe de sauvegarde revient à créer une classe MyModelSaver qui hérite de TModelSaver et qui implémente la méthode «writeData». Cette méthode reçoit en paramètre le TContainer à sauver. Elle écrit directement dans le flux de sortie «**outStream**» Ca peut-être une socket, un fichier … (dans ARTStudio c'est un fichier). Voir en fin un exemple d'utilisation de TModelSaver.

TModelParser TModelParser est une classe abstraite, implémentant la lecture d'un modèle ARTStudio (TModel). Le fichier doit être au format texte, en utilisant une structure de type XML.

Implémenter sa propre classe de lecture revient à créer une classe MyModelParser qui hérite de TModelParser et qui implémente les méthodes « handleStartTag » et « handleEndTag ».Ces méthodes sont appelées respectivement au début et à la fin de chaque tag XML. Elles reçoivent en paramètre les éléments codés dans le tag.

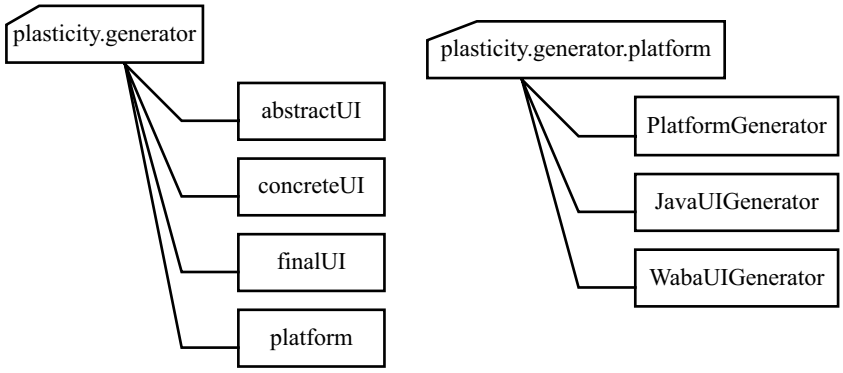
Par exemple en écrivant : <MyElem id="120"> handleStartTag reçoit une structure contenant le nom du tag (MyElem), le non de l'attribut (id) et sa valeur (120). Voir en fin un exemple d'utilisation de TModelParser.

TModel TModel est une classe abstraite. Implémenter un TModel nécessite la création d'une classe (MyModel) héritant de TModel et implémentant les méthodes getNewContainer, loadFromFile et saveToFile.

Voir un exemple d'utilisation de TModel en fin d'annexe.

2.2. LE PACKAGE « GENERATOR»

Le package des générateurs (plasticity.generator) se divise en trois sous packages. Respectivement pour la génération de l'IHM abstraite (plasticity.generator.abstractUI), la génération de l'IHM concrète (plasticity.generator.concreteUI) et de l'IHM finale (plasticity.generator.finalUI) :



Le dernier package contient les compilateurs spécifiques à chaque plate-forme (plasticity.generator.plateform). Ils traduisent l'IHM finale en code source compilable.

Aujourd'hui les plates-formes Waba et Java/JFC sont supportées. C'est le « PlateformGenerator » qui utilise le générateur adéquat pour produire le code final.

Les générateurs représentent le cœur du moteur d'inférence pour la réification. Cette inférence est implémentée selon trois méthodes :

- Soit impérative : la réification est codée en java ;
- Soit déclarative : la réification est codée sous forme de règles qui sont utilisées par le système expert JESS pour produire les modèles ;
- Soit hybride : en utilisant les deux méthodes précédente en même temps.

Génération de l'IHM abstraite

Ce générateur est codé dans la classe "plasticity.generator.abstractUI.AbstractUIGenerator".

C'est un générateur purement déclaratif. L'objet initialise le système expert puis lance l'inférence. Pour changer ou rajouter des règles de réification, il faut modifier le fichier « genAbstractUI.clp » qui se trouve dans le répertoire « jess » à la racine des exécutables.

La première partie du fichier « genAbstractUI.clp » déclare les classes manipulées par les règles de génération, les patrons des assertions et les fonctions transformant l'arbre de tâches en un ensemble d'assertions.

La deuxième partie du fichier déclare les règles de génération. Se reporter au fichier pour plus d'information.

Génération de l'IHM concrète

Ce générateur est codé dans la classe « plasticity.generator.concreteUI.GenerateConcretUI». C'est un générateur hybride.

GenerateInstance :
Génère la table de toutes les instances de concepts dans l'arbre de tâches, et leur porté. Pour chaque tâche, elle détermine les concepts externes

Génération de l'IHM finale

Ce générateur est codé dans la classe « plasticity.generator.finalUI.GenerateFinalUI». C'est un générateur purement impératif.

3. Exemples

Dans ce dernier paragraphe, nous présentons des exemples d'utilisation de code pour la création d'un modèle dans ARTStudio.

3.1. SAUVEGARDE

Pour réaliser la sauvegarde d'un modèle, il faut créer une classe héritant de TModelSaver. Voici comment procéder :

```
import concepts.*;
import concepts.model.*;

import java.util.*;
import java.io.OutputStream.*;

public class MyModelSaver extends TModelSaver
{
    public MyModelSaver (MyModel model)
    {
        super (model);
    }

    protected void writeData (TContainer container)
    {
        MyElem elem ;
        Enumeration enumElem ;

        // Récupération de la liste des éléments contenus dans le container
        Hashtable listElems = ((MyModel)theModel).getElems (container);

        // Enumération des éléments
        for (enumElem  = listElems.getElements () ; enumElem.hasMoreElements () ;)
        {
            elem = (MyElem) enumElem.nextElement ();

            // Ecriture d'un élément
            outStream.println ("<MyElem id=\"" + elem.getID () + "\">");
            outStream.println ("<name>" + elem.getName () + "</name>");
            outStream.println ("</MyElem > ") ;
        }
    }
}
```


Instanciation et utilisation de la classe :

```
MyModelSaver saver = new MyModelSaver (myModel);
saver.save (pathDir);
```

3.2. LECTURE

Pour réaliser la lecture d'un modèle, il faut créer une classe héritant de TModelParser. Voici comment procéder :

```
import concepts.*;
import concepts.model.*;

import java.util.*;
import com.ibm.xml.parser.*;

public class MyModelParser extends TModelParser
```

```
{
    MyElem currentElem;

    public MyModelParser ()
    {
        super ();
    }

    public MyModelParser (MyModel model)
    {
        super (model);
    }

    public void handleStartTag (TXElement e, boolean empty)
    {
        // Appel du handleStartTag pere.
        super.handleStartTag (e, empty);
        String n = e.getName();

        // Gestion des tags XML
        if (n.equals ("MyElem")) MyElemHandelEnd (e);
    }

    public void handleEndTag (TXElement e, boolean empty)
    {
        // Appel du handleStartTag père.
        super.handleEndTag (e, empty);
        String n = e.getName();

        // Gestion des tags XML
        if (n.equals ("MyElem")) MyElemHandel (e);
    }

    protected void MyElemHandel (TXElement e)
    {
        // Instanciation d'un MyElem
        currentElem = new MyElem () ;

        // Initialisation de l'élément
        currentElem.setID (e. getAttribute ("id") ) ;
    }

    protected void MyElemHandelEnd (TXElement e)
    {
        // A la fin du tag, l'élément est ajouté au modèle
        ((MyModel) theModel).addElem (currentElem, m_currentContainer);
    }
}
```


Instanciation et utilisation de la classe :

```
MyModelParser parser = new MyModelParser (myModel);
parser.parse (pathDir);
```

3.3. UN MODÈLE

Enfin pour réaliser un modèle, il faut créer une classe héritant de TModel. Voici comment procéder :

```
import concepts.*;
import concepts.model.*;
```

```
import java.util.*;

public class MyModel extends TModel
{
    public MyModel ()
    {
        super ();
    }

    public MyModel (TString name)
    {
        super (name);
    }

    public TContainer getNewContainer ()
    {
        // Création du container
        TContainer result = super.getNewContainer ();

        // Création des éléments contenus dans le container.
        // Dans cet exemple, le container contient une table de hachage
        // appelée « listElem »
        result.elements.put ("listElem", new Hashtable ());

        return result;
    }

    public boolean loadFromFile (TString pathDir)
    {
        // supression de tous les container dans le model
        removeAllContainer ();

        // Instanciation du parser
        MyModelParser loader = new MyModelParser (this);

        // Lecture
        loader.parse (pathDir);

        return true;
    }

    public boolean saveToFile (TString pathDir)
    {
        // Instanciation du sauveur
        MyModelSaver saver = new MyModelSaver (this);

        // Sauvegarde
        saver.save (pathDir);

        return true;
    }

    public void clear ()
    {
        super.clear ();
        removeAllContainer ();
    }

    public void addElem (MyElem elem, TContainer container)
    {
```

```
            // Ajout d'un élément à un container

            // Récuperation de la table de hachage contenant les éléments (« listElem »)
            Hashtable listElem = container.elements.get ("listElem") ;

            // Ajout dans cette table de l'élément.
            // Chaque élément a une ID qui lui sert d'identifiant unique et
            // ainsi de clef dans les tables de hachage.
            listElem .put (elem.getID (),elem);
        }

        public void addElem (MyElem elem)
        {
            // Ajout d'un élément au container courrant
            return addElem (elem, currentContainer) ;
        }

        public Hashtable getElems (TContainer container)
        {
            // Retourne les  éléments contenus dans le container
            return container.elements.get ("listElem") ;
        }

        public Hashtable getElems ()
        {
            // retourne les éléments du container courrant
            return getElems (currentContainer) ;
        }
    }
```

Annexe 3 : MMS

Avec ARTStudio, nous proposons un outil de génération d'interfaces multicibles. Cet outil permet aujourd'hui la production d'interfaces précalculées (cf. chapitre 2, page 27) : il n'est pas capable de générer des interfaces faisant de l'adaptation dynamique. Pour combler cette lacune, nous avons travaillé sur une architecture logicielle couplée à un système de résolution de contraintes, permettant une adaptation dynamique. L'objectif à long terme est de définir une architecture générique pour l'adaptation dynamique, qui serait intégrable au niveau du générateur de l'IHM Finale d'ARTSTudio. Cette annexe présente ce premier travail réalisé dans le démonstrateur MMS [Thevenin 2001].

Cette annexe s'organise comme suit :

- présentation de l'architecture pour l'adaptation dynamique ;
- présentation de MMS ;
- présentation du moteur d'inférence et de nos résultats ;
- conclusion.

1. Adaptation dynamique

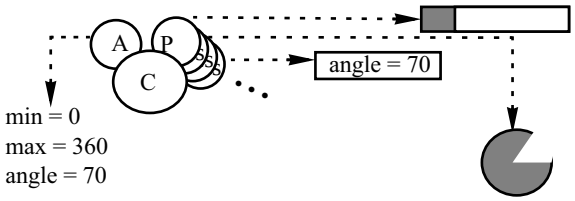
Notre approche s'appuie sur le modèle à agents PAC [Coutaz 1990] pour la description de l'interface. Ce modèle offre l'avantage de structurer l'interface en éléments indépendants (les agents PAC), et de circonscrire la partie présentation d'un agent à travers la facette Présentation (P) d'un agent. Voyons comment ces deux aspects vont nous aider à mettre en oeuvre notre interface adaptative en proposant un agent à plusieurs facettes P et une adaptation contrôlée par un médiateur.

L'agent à multifacette P

Pour l'adaptation de la présentation, nous allons adapter la représentation graphique d'une donnée. Par exemple, un angle sera représenté par

un camembert, une valeur et son unité, ou encore par une barre. Aussi l'agent PAC chargé d'afficher cette donnée possédera plusieurs présentations possibles et changera dynamiquement de présentation. Cette adaptation est possible parce que la présentation d'un agent est délimitée par sa facette P. La figure 1 montre un exemple de cet agent a multifacette P.

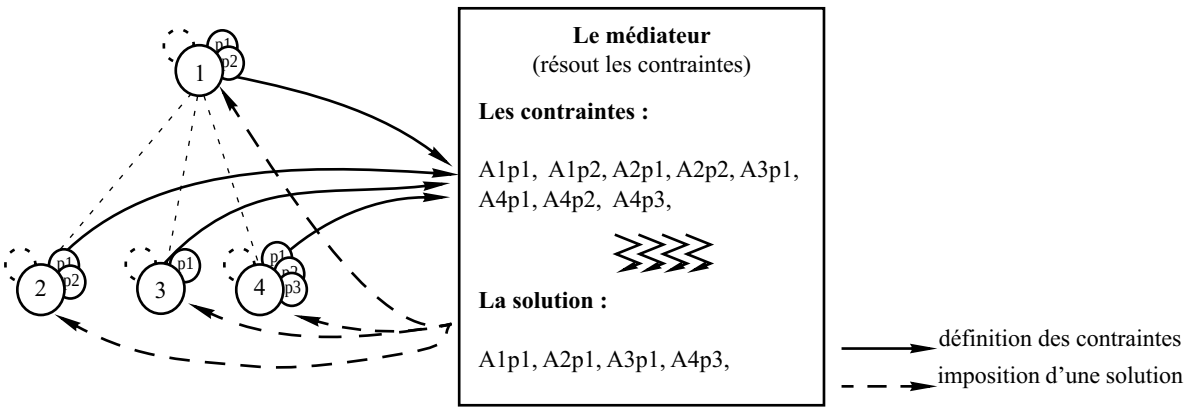
Figure 1. Agent PAC avec plusieurs présentations. Une seule est visible à un moment donné.



Le médiateur Pour la résolution des contraintes, nous avons choisi un système de médiateur. Il calcule la "meilleure" interface graphique, en fonction des contraintes de présentation de chaque agent et de la surface d'affichage disponible. Lorsqu'il a résolu le système de contraintes, il reconfigure la présentation de chaque agent. Cette reconfiguration est facilitée par l'indépendance des agents entre eux.

Chaque agent est capable de récupérer les contraintes liées à ses différentes présentations. Il les donne au médiateur qui va leur imposer une présentation. Le résolveur de contraintes est le seul à avoir la connaissance de tous les agents et donc peut résoudre le système global. La figure 2 présente les relations entre les agents et le médiateur.

Figure 2. Architecture pour la résolution des contraintes. La résolution est gobale : c'est le médiateur qui a la connaissance sur toute l'interface et qui solutionne le problème.



2. L'application MMS

MMS, pour Minimal MediaSpace, est un système qui permet de connaître le taux d'activité des personnes connectées au collecticiel. Pour les besoins de notre démonstrateur, ce taux est simulé. Mais il pourrait représenter des statistiques calculées sur la valeur numérique d'un certain nombre de capteurs — utilisation du clavier, de la souris — qui représenteraient l'activité d'une personne devant sa machine.

Les données

MMS permet donc de savoir qui est connecté, sur quelle machine et son taux d'activité. Aussi pour chaque personne, MMS représente deux chaînes de caractères (nom et machine) et un entier prenant sa valeur entre 0 et 100 (taux d'activité).

Présentations des données

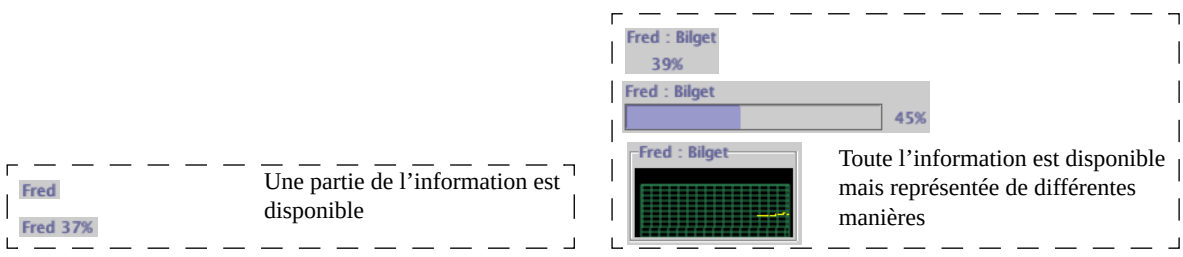
Nous avons introduit dans cette thèse le niveau d'importance d'un concept du domaine vis-à-vis de la réalisation d'une tâche dans laquelle il intervient. Dans notre cas, nous avons choisi de façon purement arbitraire — c'est aux ergonomes que revient ce délicat travail — l'ordonnancement suivant :

- le nom de la personne connectée. Le but d'un mediaspace est de donner une conscience du groupe. Aussi le nom des personnes connectées nous semble être l'information primordiale ;
- le taux d'activité. Un deuxième objectif d'un mediaspace est de donner le niveau d'activité des gens pour savoir s'il est possible de les déranger par exemple.
- enfin donner le nom de la machine sur laquelle une personne est connectée.

Aussi les données sont représentables de cinq manières différentes (cf. figure 3) regroupables en deux ensembles en fonction de l'ordonnancement précédent :

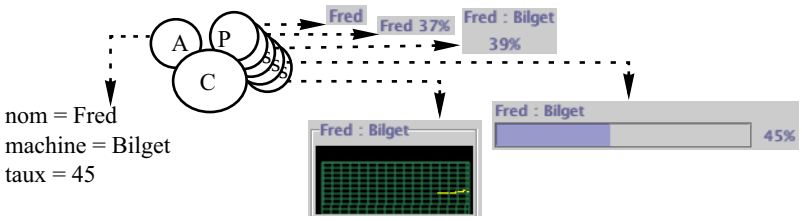
- les représentations qui affichent une partie de l'information ;
- les représentations qui affichent toute l'information mais de différentes façon.

Figure 3. Cinq interacteurs différents pour repésenter les mêmes données.



La figure 4 présente l'agent PAC chargé de présenter les informations dans MMS.

Figure 4. Agent PAC pour la représentation des données dans MMS.



3. *Inférence*

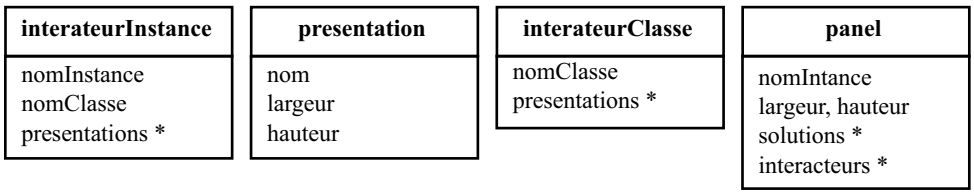
Pour la mise en oeuvre du médiateur, nous avons utilisé le système expert JESS (*Java Expert System Shell*)[JESS]. Nous allons voir dans cette section, une description des faits et des règles utilisés dans JESS. Enfin nous verrons un exemple d'inférence.

3.1. FAITS

La figure 5 présente les modèles pour les faits. Les contraintes sont déclarées, en suivant ces modèles, sous la forme de faits (ou assertions) :

- Le panel représente une interface composée d'interacteurs. Le champ "interacteurs" est la liste des instances d'interacteur à afficher, le champ "solutions" est la liste des présentations d'interacteurs pour cette interface, le champ "nom" est l'identificateur de l'interface, et les champs "hauteur" et "largeur" représentent la surface d'affichage nécessaire à cette interface ;
- Une présentation correspond à la représentation graphique d'une donnée. Elle a un nom et des contraintes de surface d'affichage ;
- Une classe d'interacteur correspond à un agent. Elle a un nom et une liste de présentation possible pour cet interacteur ;
- Une instance d'interacteur est l'instance d'un agent. Elle a un nom, un nom de la classe instanciée, et une liste de présentation pour cette instance. Cette liste est une sous-liste de la liste des présentations possibles de sa classe.

Figure 5. Modèles pour les assertions dans JESS.



3.2. RÈGLES

L'algorithme de résolution du problème est énumératif. Il construit toutes les interfaces possibles en fonction des contraintes posées par les différents agents. Des *cut* (au sens de prolog) sont faits dans l'arbre des

solutions lorsqu'un nœud ou une feuille ne respecte pas la contrainte de surface maximale d'affichage ou de cohérence. Finalement il produit un ensemble de solutions. Si l'ensemble est vide alors une présentation par défaut est choisie, sinon c'est la solution de surface d'affichage maximum qui est choisie. L'inférence fonctionne comme suit (la table 1 présente les règles d'inférence) :

- Initialement aucune interface n'a été trouvée pour un panel (le champ "solutions" est vide) ;
- Le système énumère toutes les solutions possibles d'interface pour le panel. Pour chaque interacteur du champ "interacteurs", il construit un nouveau panel par présentation possible. Dans ces nouveaux panels, il ajoute la présentation dans la liste de "solutions" et retire l'interacteur traité du champ "interaceurs" (R1);
- L'inférence s'arrête lorsque tous les panels ont le champ "interacteurs" vide ;
- Si un panel a une surface d'affichage supérieur à la surface d'affichage disponible, alors il est supprimé (R4);
- Si dans le même panel, deux interacteurs représentant la même information ont deux présentations différentes, alors il est supprimé (R2 et R3) ;
- Finalement c'est le panel ayant la surface maximale qui est élu comme interface optimale.

	Règles
R1 Enumération (énumère tous les cas)	(interacteurInstance (nomInstance ?nom1) (presentations $?l1 ?nomPresentation $?l2)) (panel (nomInstance ?nom2) (solutions $?sol) (interacteurs ?nom1 $?l3)) (presentation (nom ?nomPresentation) (largeur ?l) (hauteur ?h)) => assert (panel (nomInstance ?nom2) (largeur (calculerHauteur ?h)) (hauteur (calculerLargeur ?l)) (solutions nomPresentation o ?sol) (interacteurs ?liste))
R2 Cohérence (1)	(interacteurClasse (presentations $?l1 ?e1 $?l2 ?e2 $?l3)) ?f <- (panel (solutions $?l4 ?e1 $?l5 ?e2 $?l6)) => retract ?f
R3 Cohérence (2)	(interacteurClasse (presentations $?l1 ?e1 $?l2 ?e2 $?l3)) ?f <- (panel (solutions $?l4 ?e2 $?l5 ?e1 $?l6)) => retract ?f
R4 Surface d'affichage	?f <- (panel (largeur ?l&:(> ?l maxLargeur)) (hauteur ?h&:(> ?h maxLargeur))) => retract ?f

Table 1: Règles d'inférence de l'arbre des solutions. Les règles sont écrites au format JESS

3.3. EXEMPLE DE RÉSOLUTION

Les figures 6 et 7 montrent un exemple d'application des règles. L'inférence démarre à partir d'un ensemble d'assertions initiales, pour pro-

duire des assertions intermédiaires puis finales selon l'arbre décrit dans la figure 7.

Figure 6. Arbre de résolution et production des assertions. F9-10-12-13 représentent les interfaces possibles

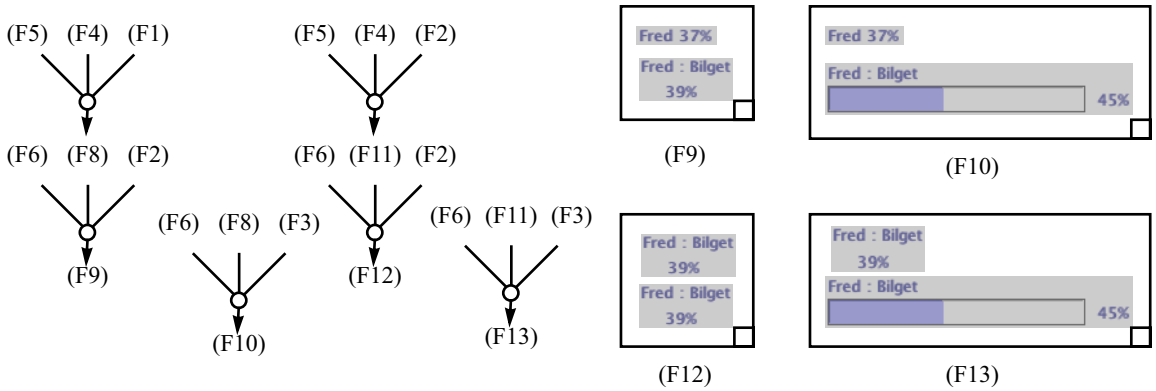


Figure 7. Exemple de résolution. Liste des assertions utilisées et produites par le résolveur de JESS

Assertions initiales :

- presentation (nom activity0) (largeur 50) (hauteur 20) (F1)
- presentation (nom activity1) (largeur 100) (hauteur 20) (F2)
- presentation (nom activity2) (largeur 50) (hauteur 50) (F3)
- panel (nomInstance MMS) (largeur 0) (hauteur 0) (solutions) (interacteurs activityJohn activityPhil) (F4)
- interacteurInstance (nomClasse PActivity) (nomInstance activityJohn) (presentations activity0 activity1) (F5)
- interacteurInstance (nomClasse PActivity) (nomInstance activityPhil) (presentations activity1 activity2) (F6)
- interacteurClasse (nomClasse PActivity) (presentations activity0 activity1 activity2) (F7)

Assertions intermédiaires :

- panel (nomInstance MMS) (largeur 0) (hauteur 0) (solutions activity0) (interacteurs activityPhil) (F8)
- panel (nomInstance MMS) (largeur 0) (hauteur 0) (solutions activity1) (interacteurs activityPhil) (F11)

Assertions finales :

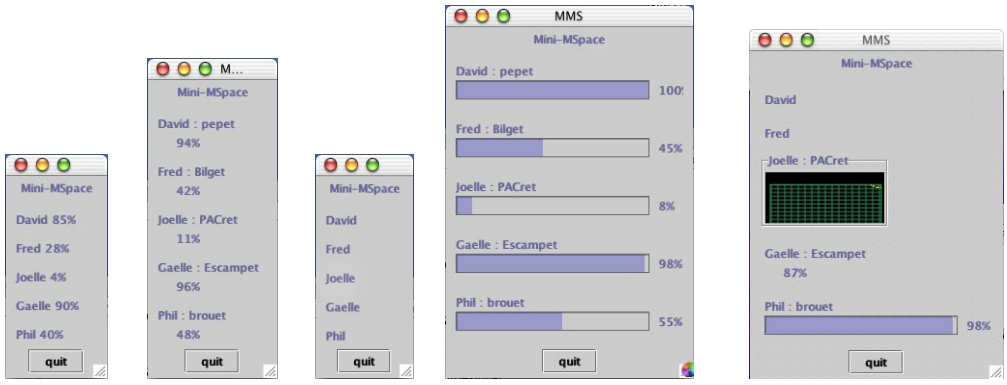
- panel (nomInstance MMS) (largeur 0) (hauteur 0) (solutions activity0 activity1) (interacteurs) (F9)
- panel (nomInstance MMS) (largeur 0) (hauteur 0) (solutions activity0 activity2) (interacteurs) (F10)
- panel (nomInstance MMS) (largeur 0) (hauteur 0) (solutions activity1 activity1) (interacteurs) (F12)
- panel (nomInstance MMS) (largeur 0) (hauteur 0) (solutions activity1 activity2) (interacteurs) (F13)

3.4. RÉSULTATS

Interfaces La figure 8 présente des interfaces produites pour MMS. Les interfaces (a) à (d) sont trouvées avec la contrainte de cohérence activée, alors que l'interface (e) est trouvée sans cette contrainte.

Temps de résolution La table 2 présente les temps de résolution en fonction du nombre de présentations par interacteur. (première ligne, 1 présentation, deuxième, 2 présentations,..., cinquième ligne, 5 présentations par interacteur). Comme le montre la figure 9b, les temps de résolution jusqu'à environ 200 interfaces ($\approx 3^5$) ne sont pas significatifs par rapport au temps perdu dans MMS pour poser les contraintes, lire le résultat, changer les interacteurs, etc. La figure 9a montre aussi un temps de résolution exponentiel, puisque le nombre d'interface est exponentiel!

Figure 8. Exemples d'interfaces produites. (a-d) avec contraintes de cohérence, (e) sans cette contrainte

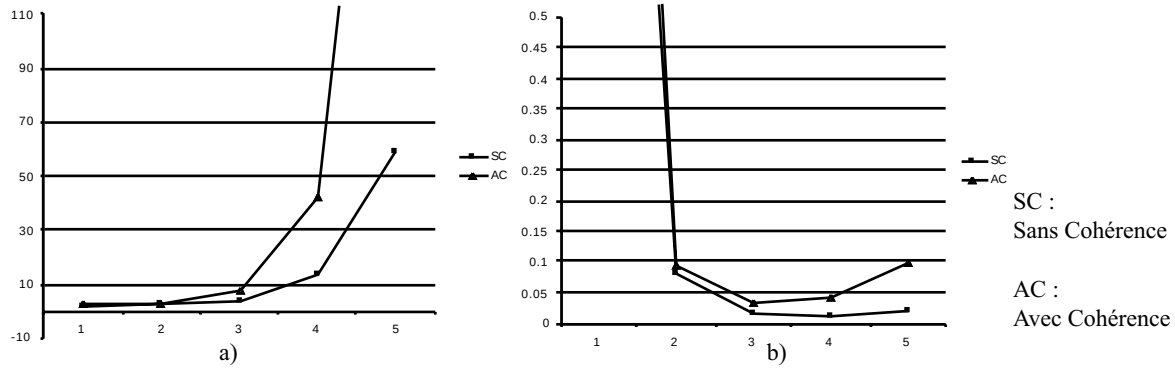


(a) (b) (c) (d) (e)

Nombre d'interface possibles	sans cohérence	avec cohérence
$1^5 = 1$	2 s	2,3 s
$2^5 = 32$	2,6 s	3 s
$3^5 = 243$	3,9 s	7,7 s
$4^5 = 1024$	13,9 s	42,3 s
$5^5 = 3125$	59 s	> 300 s

Table 2: Temps de résolution

Figure 9. Courbes de comparaison des temps. Fig a) courbes des temps b) courbes des temps divisé par le nombre d'interface.



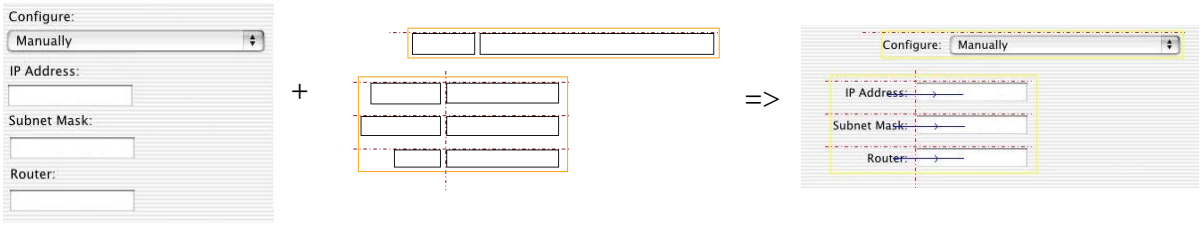
4. Conclusion

L'architecture proposée semble bien fonctionner. Dynamiquement l'interface change sa présentation avec un "minimum d'effort" de pro-

grammation. La principale limite du système, outre le temps de résolution (possiblement améliorable en faisant de la propagation de contraintes), est la gestion du placement des interacteurs.

Ce placement est réalisé par une fonction ad hoc qui positionne les interacteurs verticalement. Nous voudrions étendre cette fonction pour gérer d'autres types de placement en utilisant des contraintes sur le positionnement entre interacteurs — du type : interacteur X est à gauche du Y. A partir de ces contraintes de placement il serait possible de définir un modèle de présentation (cf. figure 10) qui devrait être conservé lors de l'adaptation.

Figure 10. Exemple d'interface avec un modèle de présentation.



Interface avec un modèle de présentation verticale

Modèles de présentation

Interface finale

Bibliographie

1. Références littéraires

[Akoumianakis et Stephanidis 1997]
D. Akoumianakis et C. Stephanidis, Supporting user-adapted interface design: The USE-IT system. Paru dans le journal Interacting with Computer n°9, chez Elsevier, 1997, pp. 73-104.

[Anderson et al. 2000]
G. Anderson, N. Graham, T. Wright, "Dragonfly: Linking Conceptual and Implementation Architectures of Multiuser Interactive Systems", Actes de la conférence ICSE 2000, publication ACM, pp. 252-26, 2000.

[André et al. 1993]
E. André, W. Finkler, W. Graf, T. Rist, A. Schauder et W. Wahlster, WIP: The Automatic Synthesis of Multimodal Presentations. Paru dans [Roth et Hefley 1993], pp. 75-93.

[Arens et Hovy 1995]
Y. Arens et E. Hovy, The Design of a Model-Based Multimedia Interaction Manager. Paru dans *Artificial Intelligence Review*, v°9, n°2-3,1995, pp.167-188.

[Badros 2000]
G. J. Badros. *Extending Interactive Graphical Applications with Contraints*. Thèse pour l'obtention du titre de docteur en informatique, Université de Washington, 2000.

[Badros et al. 2000]
G. J. Badros, J. Nichols, A Borning, SCWM: An Intelligent Constraint-Enabled Window Manager, American Association for Artificial Intelligence, mars, 2000.

[Balbo 1994]
S. Balbo, Evaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation. Thèse pour l'obtention du tire de docteur en informatique, Université Joseph Fourier, Grenoble I, septembre, 1994.

[Bardram 1997]
J. E. Bardram, Plans as Situated Action: An Activity Theory Approach to Workflow Systems. Actes de la conférence ECSCW'97, Lancaster UK, Septembre 1997, pp. 17-32.

[Bass 1991]
L. Bass, R. Little, R. Pellegrino, S. Reed, R. Seacord, S. Sheppard, The Arch Model: Seeheim Revisited (version 1.0). The UIMS Tool Developers Workshop (Avril 1991) in ACM SIGCHI Bulletin Vol. 24, No. 1, Janvier 1992.

[Berthomé-Montoy 1995]
A. Berthomé-Montoy, Une approche descriptive de l'auto-adaptativité des interfaces homme-machine. Thèse pour l'obtention du titre de docteur en informatique, Université Claude Bernard, Lyon, 1995.

[Bodart et Vanderdonckt 1994]
F. Bodart, j. Vanderdonckt, On the Problem of Selecting Interaction Objects. Actes de la conférence *Human Computer Interaction* (HCI'94), People and Computer IX, 1994, pp. 163-178.

[Brisson et Andre 94]
G. Brisson et J. Andre, PROSPECT. Analyse et spécification de l'interface utilisateur d'un système interactif. Actes du colloque ERGO.IA'94, Biarritz, 1994, pp. 382-393.

[Browne et al. 1990a]
D. Browne, P.Totterdell, M. Norman, Adaptive User Interfaces. Paru chez Acadamic Press dans la collection Computer And People Series, 1990.

[Browne et al. 1990b]
D. Browne, M. Norman, d. Riches, Why Build Adaptive Systems? Dans [Browne et al. 1990a], pp. 15-58.

[Bruley 1999]
C. Bruley, Analyse des représentations graphiques de l'information - extension aux représentations tridimensionnelles. Thèse pour l'obtention du titre de docteur en informatique, Universite J. Fourier, Grenoble I, France, 1999.

[Brun 1998]
P. Brun, XTL : une logique temporelle pour la spécification formelle des systèmes interactifs. Thèse pour l'obtention du titre de docteur en informatique, Université Paris XI Orsay, 1998.

[Bruno 1995]
G. Bruno, Model Based Software Engineering. Editer chez Chapman & Hall, 1995. ISBN 0 412 48670 9.

[Brusilovsky 2001]
P. Brusilovsky, Adaptive Hypermedia. Paru dans le journal User Modeling and User-Adapted Interaction 2001, (AMAUI'01), vol. 11, n° 1-2, edition spéciale pour le trentième aniversaire, éditeur A. Kobsa, publié chez Kluwer Academic Publishers, pp. 87-110.

[Buxton 1983]
W. Buxton, Lexical and pragmatic consideration of input structures. Computer Graphics v.17, n°1, 1983, pp.31-37.

[Calvary 1998]
G. Calvary, Proactivité et Réactivité : De l'assignation à la complémentarité en conception et évaluation d'interfaces homme-machine. Thèse pour l'obtention du titre de docteur en informatique, Université Joseph Fourier - Grenoble I, 1998.

[Card 1983]
S. K. Card, T. P. Moran et A. Newell, The psychology of Human Computer Interaction, Lawrence Erlbaum Associates, 1983.

[Channac 1999]
S. Channac, Conception et mise en œuvre d'un système déclaratif de géométrie dynamique. Thèse pour l'obtention du titre de docteur en informatique, Université Joseph Fourier, Grenoble I, 1999.

[Charman 1995]
P. Charman, Gestion des contraintes géométriques pour l'aide à l'aménagement spacial. Thèse pour l'obtention du titre de docteur en informatique, Ecole nationale des ponts et chaussées, 1995.

[Cheverst et al. 2001]
K. Cheverst, N. Davies, K. Mitchell et C. Efstratiou. Using Context as a Crystal Ball: Rewards and Pitfalls. Paru dans Personal

and Ubiquitous Computing, 5(1), Springer Verlag Publ., 2001, pp. 8-11.

[Coutaz 1990]
J. Coutaz, Interfaces homme-ordinateur, Conception et réalisation. Chez Dunod Informatique, 1990.

[Coutaz et Nigay 2001]
J. Coutaz, L. Nigay, Architecture logicielle conceptuelle des système interactifs, Chapitre 7 du livre "Analyse et Conception de l'Interaction Homme-Machine", Hermes Editeur, ISBN 2-7462-0239-5, 2001, pp. 207-246.

[Crease et al. 2000]
M. Crease, P. D. Gray, S. A. Brewster, A Toolkit of Mechanism and Context Independent Widgets. Actes du workshop Design, Specification, and Verification of Interactive Systems, DSVIS'00, 2000, pp. 121-133.

[Crow et Smith 1993]
D. Crow et B. Smith, The Role of Build-in Knowledge in Adaptive Interface Systems. Actes du workshop Intelligent User Interface (IUI'93), éditeurs W. D. Gray, W. E. Hefley, D. Murray, publié chez ACM Press, Orlando, janvier 1993, pp. 97-104.

[Dey 2000]
A. Dey, Providing Architectural Support fo Building Context-Aware Applications. Thèse pour l'obtention du titre de docteur en Informatique, Institut Technologique de Géorgie (Georgia Tech), 2000.

[Dieterich et al. 1993]
H. Dieterich, U. Malinowski, T. Kühme, M. Schneider-Hufschmidt. State of the Art in Adaptive User Interfaces. Dans [Schneider-Hufschmidt et al. 1993], pp.13-48.

[Dix et al. 1998]
A. Dix, J. Finlay, G. Abowd, R. Beale, Human-Computer Interaction (Second edition). Chez Prentice Hall Europe, 1998.

[Ducournau 1996]
R. Ducournau, Des langages à objets aux logiques terminologiques : les systèmes classificatoires. Rapport de Recherche 96-030, LIRMM, Montpellier, 1996, ftp.lirmm.fr/pub/LIRMM/papers/1996/RRI-Ducournau-96.ps.

[Eisenstein et Puerta 2000]
J. Eisenstein et A. Puerta, Adaptation in Automated User-Interface Design. Actes de la conférence internationnale Intelligent User Interfaces (IUI'00), ACM Press, Nouvelle Orléan, 2000, pp.74-81.

[Eisenstein et al. 2001]
J. Eisenstein, J. Vanderdonckt, A. Puerta, Applying Model-Based Techniques to the Development of UIs for Mobile Computers. Actes de la conférence internationnale Intelligent User Interfaces (IUI'01), ACM Press, Santa Fe, janvier 2001, pp.74-81.

[Estublier 2000]
J. Estublier, Software Configuration Management: A Road Map. The Future of Software Engineering; Ed. Anthony Finkelstein; ACM Press, May 2000, ISBN 1-58113-253-0.

[Euzenat 1998]
J. Euzenat, Sémantique des représentations de connaissance. Cours de DEA-ISC, Univerisité Joseph Fourier, Grenoble, 1998. http://exmo.inrialpes.fr/people/euzenat/.

[Feiner et McKeown 1993]
S. K. Feiner et K. R. McKeown, Automating the Generation of Coordinated Multimedia Presentations. Paru dans [Maybury 1993], pp. 117-138.

[Fekete 1996]
J. D. Fekete, Les trois services du noyau sémantique indispensables à l'IHM. Actes des journées de travail d'IHM'96, 20-22 novembre 1996, Grenoble. pp. 45-50.

[Foley 1984]
J.D. Foley, V.L. Wallace, P. Chan, The Human Factors of computer Graphics interaction techniques, IEEE computer Graphics and Applications, v.4, n°11, 1984, pp.13-48.

[Frank 1998]
M. R. Frank, P. Szekely, Adaptive Forms: An Interaction Technique for Entering Structured Data. Actes de la conférence Intelligent User Interfaces (IUI'98), ACM Press, San Francisco, Californie, 6-9 janvier, 1998, pp. 153-160.

[Gamboa et Scapin 1997]
R. F. Gamboa et D. L. Scapin, Editing MAD* task description for specifying user interfaces, at both semantic and presentation levels. Actes de *Eurographics Workshop on Design, Specification and Verification of Interactive Systems* (DSV-IS'97), Grenade, Espagne, Juin, 1997, pp. 193-208.

[Gamma et al. 1995]
E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software. Chez Addison-Wesley Professional Computing series, 1995.

[Gram et Cockton 1997]
C. Gram et G. Cockton, Design Principles for Interactive Software. UK, Chapman and Hall, 1996.

[Grolaux et al. 2001]
D. Grolaux, P. Van Roy and J. Vanderdonckt, QTk: An Integrated Model-Based Approach to Designing Executable User Interfaces. A paraître dans les actes de la conférence *Design, Specification and Verification of Interactive Systems*, (DSVIS'01), juin 2001, Glasgow.

[Halverson 1994]
C. A. Halverson, Distributed Cognition as a theoretical framework for HCI: Don't throw the Baby out with the bathwater - the importance of the cursor in Air Traffic Control. Rapport technique No. 94-03, Dept. of Cognitive Science, Université de Californie, San Diego, 1994.

[Haton et al. 1991]
J.P. Haton, N. Bouzid, F. Chapillet, M.C. Haton, B. Lâasri, H. Lâasri, P. Marquis, T. Mondot, A. Napoli, Le Raisonnement en Intelligence Artificielle, Modèles, techniques et architecture pour les systèmes à base de connaissances. Paru chez InterEditions, collection Informatique Intelligence Artificielle (IIA), Paris, 1991.

[Hix et Hartson 1993]
D. Hix et H. R. Hartson, Developing User Interfaces. Ensuring Usability Through Product & Process. Paru chez Wiley Professional Computing NY, 1993.

[Johnson et al. 1993]
P. Johnson, S. Wilson, P. Markopoulos, Y. Pycock, (1993) ADEPT-Advanced Design Environment for Prototyping with Task Model. Actes de la conférence Human Factors in Computing Systems INTERCHI'93 (association de CHI'93 et INTERACT'93), ACM Press, Amsterdam, avril 1993, pp. 56.

[Kobsa et al. 2001]
A. Kobsa, J. Koenemann and W. Pohl: Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships. A paraître dans The Knowledge Engineering Review 2001. http://www.ics.uci.edu/~kobsa/papers/2001-KER-kobsa.pdf.

[Luyten et Coninx 2001]
K. Luyten et K. Coninx, An XML-based runtime user interface description language for mobile computing devices. A paraître dans les actes de la conférence *Design, Specification and Verification of Interactive Systems*, (DSVIS'01), juin 2001, Glasgow.

[Mackinlay 1990]
J. Mackinlay, S. K. Card, G. G. Robertson, A Semantic Analysis of the Design Space of Input Devices. In Human-Computer Interaction, v.5, 1990, pp.145-190.

[Markopoulos 1995]
P. Markopoulos, On The Expression Of Interaction Properties With An Interactor Model. Actes de la conférence Design, Specification and Verification of Interactive System (DSVIS'95), 1995, éditeurs P. Palanque, R. Bastide, Chez SpringerComputerScience, pp. 294-310.

[Maybury 1993]
M. T. Maybury, Intelligent Multimedia Interfaces. Edité chez AAAI Press / The MIT Press, 1993, Menlo Park, Californie.

[Muller 1999]
P. A. Muller, Modélisation objet avec UML. Edité chez Eyrolles, 2000.

[Myers et al. 1990]
B. A. Myers, D. A. Guise, R. B. Dannenberg, B. Vander Zanden, D. S. Kosbie, E. Pervin, A. Mickish et P. Marchal. GARNET comprehensive support for graphical, highly interactive user interfaces. Publié dans IEEE Computer magazine, novembre, 1990, pp. 71-85.

[Nanard 1990]
J. Nanard, La manipulation directe en interface homme-machine. Thèse pour l'obtention du titre de Docteur d'Etat en science, Université de Montpellier II, 1990.

[Nigay 1994]
L. Nigay, Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales. Thèse pour l'obtention du titre de docteur en informatique, l'Université Joseph Fourier, Grenoble I, 1994.

[Nigay et Coutaz 1996]
L. Nigay, J. Coutaz Espaces conceptuels pour l'interaction multimédia et multimodale, TSI, spécial Multimédia et

Collecticiel, AFCET&Hermes Publ., Vol 15(9), 1996, pp. 1195-1225.

[Normand 1992]
V. Normand, Le Modèle Siroco : de la spécification conceptuelle des interfaces utilisateur à leur réalisation. Thèse pour l'obtention du titre de docteur en informatique, Université Joseph Fourier, Grenoble I, 1992.

[OMG 1991]
Object Management Group, The Common Object Request Broker: Architecture and specification, num. 91.12.1, December 1991.

[Opperman et Simm 1994]
R. Opperman et H. Simm, Adaptability: User-initiated individualization, in R. Oppermann (ed), Adaptive User Support, Lawrence Earlbaum, 1994.

[Ousterhout 1994]
J.K. Ousterhout, Tcl and the Tk Toolkit. Addison-Wesley, Reading, 1994. Documentations disponibles sur le site Internet http://www.scriptics.com/scripting/.

[Paternò et al. 1997]
F. Paternò, C. Mancini & S. Meniconi, ConcurTaskTrees: a diagrammatic notation for specifying task models. Actes de la conférence Human-Computer Interaction 1997 (INTERACT'97), Editeurs S. Howard, J. Hammond & G. Lindgaard, Chapman &Hall, pp. 362-369.

[Phanariou 2000]
C. Phanariou, UIML: a Device-Independent User Interface Markup Language. Thèse pour l'obtention du titre de docteur en informatique, Institut Polytechnique de Virginie (faculty of Virginia Polytechnic Institute and State University), Blacksburg, Virginie, septembre, 2000.

[Prolog IV]
Le manuel de prolog IV, Tutoriel, Concepts de base, Primitives, Prolog ISO, Syntaxe, Environnement. PrologIA, Parc Technologique de Luminy, Case 919, 13288 Marseille cedex 09, France.

[Puerta 1996]
A. Puerta, The Mecano Project: Comprehensive and Integrated Support for Model-based Interface Development. Actes de la de la conférence Computer-Aided Design of User Interfaces (CADUI'96), ed. J. Vanderdonckt, Namur University Press, Namur, 1996, pp.19-36.

[Quesnot 1995]
D. Quesnot, Spécification d'interfaces Homme-Machine et Prototypage. Thèse pour l'obtention du titre de docteur en informatique, Institut National Polytechnique de Lorraine, 1995.

[Renevier et Nigay 2001]
P. Renevier, L. Nigay, Mobile Collaborative Augmented Reality: the Augmented Stroll. Actes de la conférence EHCI'01, IFIP WG2.7 (13.2), Toronto, mai 2001, LNCS 2254, Spinger-Verlag, pp. 315-334.

[Rey 2001]
G. Rey, Systèmes interactifs sensibles au contexte. Rapport de DEA d'Informatique Systèmes et Communications, Université Joseph Fourier, Grenoble I, 2001.

[Roth 1990]
S. Roth, J. Mattis, Data Characterization for Intelligent Graphics Presentation. Actes de la conférence Human Factors In Computing Systems (CHI'90), ACM Press, 1990, pp. 193-200.

[Roth et Hefley 1993]
S. F. Roth, W. E. Hefley, Intelligent Multimedia Presentation Systems: Research and Principles. Paru dans [Maybury 1993] pp. 13-59.

[Savidis et Stephanidis 1998]
A. Savidis, C. Stephanidis, The HOMER UIMS for dual user interface development: Fusing visual and non-visual interactions. Paru dans le journal Interacting with Computer n°11, chez Elsevier, 1998, pp.173-209.

[Savidis et al. 2001]
A. Savidis, D. Akoumianakis et C. Stephanidis, The Unified User Interface design method. Paru dans [Stephanidis 2001], pp. 417-440.

[Schneider-Hufschmidt et al. 1993]
M. Schneider-Hufschmidt, T. Kühme, U. Mallinowski, Adaptive User Interfaces, Principles and Practice, editeurs, H-J Bullinger, p. G. Polson, dans la série Human Factors in Information Technologie 10 paru chez Elsevier Science Publishers B.V, North-Holland, 1993.

[Schuman 1987]
L. A. Schuman, Plans and situated actions. The problem of human-machine communication. Edité chez Cambridge University Press, 1987.

[Sears 1993]
A. Sears, Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout. Paru dans IEEE Transactions on Software Engineering 19 (7), pp.707–719.

[Sherman 1993]
E. H. Sherman. A User-Adaptable Interface to Predict Users' Needs. Paru dans [Schneider-Hufschmidt et al. 1993] pp. 285-316.

[Stephanidis 2001]
User Interfaces for All - concepts, methods and tools. K. Stephanidis (ed.), NJ Mahwah : Lawrence Erlbaun Associates, 2001, ISBN 0-8058-2967-9, 728 pages.

[Stephanidis et al. 2001]
C. Stephanidis, A. Paramythis, M. Sfyrakis et A. Savidis, A case Study in Unified User Interface Development: The AVANTI Web Browser. Paru dans [Stephanidis 2001] pp. 525-568.

[Stephanidis et Savidis 2001]
C. Stephanidis et A. Savidis, Universal Access in the Information Society: Methods, Tools, and Interaction Technologies. Paru dans le journal international Universal Access in the Information Society (UAIS), v°1, n°1, juin 2001, ed. C. Stephanidis, chez Springer, pp. 40-55.

[Sukaviriya et al. 1994]
N. Sukaviriya, S. Kovacevic, J. D. Foley, B. A. Myers, D. R. Olson Jr, M. Schneider-Hufschmidt, Model-Based User Interfaces: What are They and Why Should We Care? Actes de la conférence User Interface Software and Technology (UIST'94), Marina del Rey, Californie, novembre, 1994, pp. 133-135.

[Sukaviriya et Foley 1993]
P. Sukaviriya et J. D. Foley, Supporting adpative interfaces in a knowledge-based user interface environment. Actes du workshop international Intelligent User Interfaces (IUI'93), ACM Press, Orlando, janvier 1993, pp. 107-113.

[Sutcliffe 1997]
A. Sutcliffe, Task-Related Information Analysis. Paru dans le journal international Human-Computer Studies, n°47, 1997, pp. 223-257

[Szekely 1996]
P. Szekely, Retrospective and challenges for Model-Based Interface Development. Actes de la conférence Computer-Aided Design of User Interfaces (CADUI'96), ed. J. Vanderdonckt, Presses Universitaires de Namur, 1996, pp. xxi-xliv.

[Tarby 1993]
J.C. Tarby, Gestion Automatique du Dialogue Homme-Machine à partir de Spécifications Conceptuelles. Thèse pour l'obtention du titre de docteur en informatique, Université de Toulouse I, 1993.

[Thevenin 1998]
D. Thevenin, Interfaces Homme Machine Autoconfigurables. Rapport de DEA d'Informatique, Université Joseph Fourier, Grenoble I, 1998.

[Thevenin et Coutaz 1999]
D.Thevenin, J. Coutaz, Plasticity of User Interfaces: Framework and Research Agenda. Dans les actes de la septième conférence IFIP *on Human Computer Interaction*, Interact'99, Edinburgh, Ecosse, 1999, pp. 110-117.

[Tim Burton 1998]
Tim Burton, La triste fin du petit Enfant Huître et autres histoire. U.G.E. Poche Editions, 1998.

[Totterdell et Rautenbach 90]
P. Totterdell, P. Rautenbach, Adaptation as a Problem Design, in Adaptive User Interfaces. Paru dans [Browne et al. 1990a], pp. 59-84.

[Vanderdonckt 1997]
J. Vanderdonckt, Conception assisté de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive. Thèse pour l'obtention du titre de docteur en informatique, Facultés Universitaires Notre-Dame de la Paix, 1997.

[Vanderdonckt et Bodard 1993]
J. Vanderdonckt et F. Bodard, Encapsulating Knowledge For Intelligent Automatic Interaction Objects Selection. Actes de la conférence Human Factors in Computing Systems INTERCHI'93 (association de CHI'93 et INTERACT'93), ACM Press, Amsterdam, avril 1993, pp. 424-429.

[Vanderdonckt et al. 2001]
J. Vanderdonckt, L. Bouillon, et N. Souchon. Flexible reverse engineering of web pages with VAQUISTA. A paraître dans les actes de IEEE 8th Working Conference on Reverse Engineering (Stuttgart, 2-5 Octobre 2001), Los Alamitos, 2001. IEEE Press.

[Vernier 2001]
F. Vernier, La multimodalité en sortie et son application à la visualisation de grande quantités d'information. Thèse pour

l'obtention du titre de docteur en informatique, Université Joseph Fourier, Grenoble I, 2001.

[Weiser 1993]
M. Weiser, Some Computer Science Problems in Ubiquitous Computing, Communications of the ACM, Juillet 1993. Republié dans "Ubiquitous Computing". Nikkei Electronics, Décembre 1993, pp. 137-143.

[Wise et Glinert 1995]
G. B. Wise et E. P. Glinert, MetaWidgets for Multimodal applications. Actes de la conférence RESNA'95, Vancouver, Canada, 1995, pp. 455-457.

[Zizi et Beaudouin-Lafon 1994]
M. Zizi, M. Beaudouin-Lafon, Accessing Hyperdocuments through Interactive Dynamic Maps. Actes de la conférence European Conference on Hypermedia Technology (ECHT'94), ACM Press, 1994, pp.126-135.

2. Références sur Internet

[Abrams et Phanariou 1999]
M. Abrams et C. Phanariou, UIML: An XML Language for Building Device-Independent User Interfaces. XML '99, Dec. 1999, Philadelphie. http://www.xml.com/pub/a/1999/12/xml99/wrapup.html. Article disponible sur http://www.harmonia.com/resources/papers/index.htm.

[ArgoUML]
Outils pour la conception orientée objet. Outils et documentation disponibles sur Internet : http://argouml.tigris.org/.

[Biron 2001]
P. Biron, A. Malhotra, XML Schema Part 2 : Datatypes, 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[Bray et al. 2000]
T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/2000/REC-xml-20001006/.

[Clark 1999]
J. Clark, XSL Transformations (XSLT), 1999, http://www.w3.org/TR/1999/REC-xslt-19991116/.

[Clark 2000]
D. Clark, From Abstract to Concrete: designing AUIML renderers. White Paper, mai 2000, http://www-3.ibm.com/ibm/easy/eou_ext.nsf/EasyPrint/878.

[CLIPS]
CLIPS: *C Language Integrated Production System*. http://www.ghg.net/clips/CLIPS.html

[Grolaux 2000]
D. Grolaux, FlexClock. Disponible sur le site Internet http://www.info.ucl.ac.be/people/ned/flexclock/.

[GTK]
http://www.gtk.org/

[IBM OCL]
The Object Constraint Language OCL, the expression language for the UML. http://www-4.ibm.com/software/ad/library/standards/ocl.html.

[JESS]
Java Expert System Shell : http://herzberg.ca.sandia.gov/jess/.

[Merrick 2001]
R. A. Merrick, Device Independent User Interfaces in XML. BelCHI 2001. http://www.belchi.be/event.htm.

[Mozart]
The Mozart Programming System. http://www.mozart-oz.org/

[SUN/J2SE]
http://java.sun.com/j2se/1.3/docs/

[Thevenin 2001]
Publié dans [Thevenin et Coutaz 1999] et disponible sur le site Internet http://iihm.imag.fr/thevenin/demos/MMS/MMS.en.html.

[Thomson et al. 2001]
H. Thomson, D. Beech, M. Maloney, N. Mendelsohn, XML Schema Part 1: Structures, 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

Table des auteurs

Définitions

A adaptabilité

L'adaptabilité (adaptability) est la propriété caractérisant une interface utilisateur adaptable. 162,
Pour [Stephanidis et al. 2001] l'adaptabilité caractérise une interface capable de s'adapter en utilisant les connaissances déterminées avant l'exécution. Cette adaptation est automatique. 162

adaptable

Une interface utilisateur est dite adaptable (qui peut être adaptée) lorsqu'elle peut-être modifiée par l'utilisateur selon ses besoins [Opperman et Simm 1994]. 162

adaptative

Une interface utilisateur adaptative (adaptive user interface) est une interface capable de changer son comportement automatiquement pour correspondre à un individu ou groupe d'individus [Browne et al. 1990b]. 162

adaptativité

L'adaptativité (adaptativity) est la propriété caractérisant une interface utilisateur adaptative. 161,
Pour [Stephanidis et al. 2001] l'adaptativité caractérise une interface capable de capturer les informations pour son adaptation, en cours d'exécution. Cette adaptation est automatique. 163

C cible

Une cible se définit par le triplet <classe d'utilisateurs, plate-forme, environnement > 3

consommation

La consommation d'une décoration d'un élément de description est

R reconfigurabilité

La reconfigurabilité est la capacité du système à supporter un ajustement, une modification de l'interaction en entrée ou en sortie, fait par l'utilisateur [Gram et Cockton 1997]. 162

réutilisabilité des données d'entrée et de sortie

La réutilisabilité des données d'entrée et de sortie est la capacité de l'interface, de réutiliser des données automatiquement : les sorties du système peuvent être utilisées comme des données d'entrée (par exemple le couper-coller), ou les entrées de l'utilisateur peuvent être réutilisées par le système en sortie (par exemple les valeurs par défaut). 163

S souplesse

La souplesse (flexibility) caractérise ce qui s'adapte aisément aux circonstances. [Gram et Cockton 1997] définissent cette propriété comme la capacité du système à offrir plusieurs chemins entre l'utilisateur et le système pour l'échange d'information lors de l'interaction. Elle exprime l'éventail des choix (pour l'utilisateur et le système) [Coutaz et Nigay 2001]. 161

sous-contraint

Un problème est dit sous-contraint lorsque l'ensemble des contraintes définies n'est pas suffisant pour que la résolution du système donne une unique solution. 131

sur-contraint

Un problème est dit sur-contraint lorsqu'aucune solution ne peut satisfaire l'ensemble des contraintes posées. 131

T tâche élémentaire

Une tâche élémentaire est la plus petite tâche d'interaction indépendante de la plate-forme. Si elle devait être décomposée plus avant, on obtiendrait des tâches d'interaction dépendantes de la cible. 107

RÉSUMÉ

Cette thèse s'inscrit dans le domaine de l'Ingénierie de l'Interaction Homme-Machine. Elle traite de la production d'Interfaces Homme-Machine (IHM) multicible. Une IHM multicible a des capacités d'adaptation à plusieurs cibles tout en respectant l'utilisabilité. Une cible se définit par le triplet <plate-forme, environnement, utilisateur>. La plate-forme désigne le support matériel et logiciel qui sous-tend l'interaction. L'environnement dénote le milieu dans lequel s'exerce l'interaction. La conception et le développement d'IHM multicible présentent des difficultés majeures de Génie Logiciel : partage de ressource, communication et maintenance entre les différents projets, explosion sous l'effet de la combinatoire.

En réponse à ces problèmes, nous analysons les taxonomies et les outils existants en rapport avec l'adaptation. De cette étude, nous proposons un espace de classification qui met en évidence les lacunes des outils existants. Ce constat conduit à de nouvelles propositions en matière d'outils de spécification d'IHM multicible. Nous énonçons un ensemble de principes fondamentaux : la factorisation et la décoration qui militent en faveur de la capitalisation de connaissances. Ces deux principes s'inscrivent dans un cadre théorique qui, fondé sur la coopération des principes de réification et de traduction, constitue un processus de référence conceptuel pour la définition d'outils de production d'IHM multicible.

Nous réalisons le processus de référence pour le cas de la génération d'IHM plastiques, c.-à-d. d'IHM capables d'adaptation à plusieurs plates-formes et environnements. Avec ARTStudio et MMS, nous concrétisons notre processus de référence. ARTStudio est un générateur d'IHM plastique qui permet au concepteur d'intervenir à tous les niveaux du processus de production. MMS montre la réalisation de mécanismes logiciels permettant l'adaptation dynamique. Dans leur version actuelle, ARTStudio et MMS ne couvrent que l'adaptation à la surface d'affichage.

MOTS-CLÉS : Interaction Homme-Machine, Interface Utilisateur, Adaptation, Plasticité, Conception, Multicible, Générateur, ARTStudio

TITRE EN ANGLAIS : Adaptation in Human Computer Interaction: the case of Plasticity

RÉSUMÉ EN ANGLAIS

This thesis contributes to the Software Engineering domain of Human Computer Interaction. It addresses the development of multi-target User Interfaces (UI), i.e., UI's that can adapt to multiple targets while preserving usability. A target is defined by a triple "platform, environment, user", where the platform denotes the software and devices that underlie interaction, and the environment corresponds to the physical space where the interaction takes place. Developing a multi-target UI involves a panoply of software engineering problems including resource sharing, versioning, and the combinatory explosion due to the variety of potential targets.

In response to these problems, this dissertation begins with an analysis of the taxonomies and tools that address the problem of adaptation. This study leads to the proposal of a classification space that identifies the weaknesses of current tools. We then propose a general framework that structures the development process of multi-target UI's. This framework, which adopts a model-based approach, conveys the following four principles: factorization, decoration, reification and translation. Factorization and decoration are used as complementary means for supporting specification reuse. Reification and translation are used in conjunction to derive executable multi-target UI's from high-level specifications.

Our framework has been used for the development of Plastic User Interfaces. A Plastic UI is a kind of multi-target UI whose adaptation is limited to platforms and environments. This framework has been instantiated within two tools: ARTStudio and MMS. ARTStudio is a semi-automatic generator of plastic UI's that provides the designer with the ability to tune the descriptions generated by the system at every step of the reification process. In its current version, ARTStudio addresses variations of screen size only. MMS is a mediaspace end user application that can dynamically adapt to screen size changes.

MOTS-CLÉFS EN ANGLAIS : Human Computer Interaction, User Interface, Adaptation, Plasticity, Multi-target, Generator, ARTStudio

DISCIPLINE : Interaction Homme Machine

UNIVERSITÉ JOSEPH FOURIER - GRENOBLE 1
LABORATOIRE COMMUNICATION LANGAGIÈRE ET INTERACTION PERSONNE-SYSTÈME