Modèles et outils pour la conception et l'exécution d'Interfaces Homme-Machine Plastiques, Ecosystème

Alexandre Demeure

THESE
présentée par
Alexandre Demeure

Pour obtenir le titre de
DOCTEUR de L'UNIVERSITE JOSEPH FOURIER - GRENOBLE I
(Arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)
Spécialité : **Informatique**

Modèles et outils pour la conception et l'exécution d'Interfaces Homme-Machine Plastiques

Date de soutenance : 11 Octobre 2007

Composition du jury :

Directeur de thèse :	Joëlle Coutaz
Co-directeur de thèse :	Gaëlle Calvary
Présidente :	Catherine Berrut
Rapporteurs :	Philippe Palanque
	Jean Vanderdonckt
Examinateurs :	Eric Lecolinet
	Jean-Claude Tarby

Thèse préparée au sein du Laboratoire d'Informatique de Grenoble (LIG)

Remerciements

Je tiens tout d'abord à remercier Sabrina de m'avoir soutenu tout au long de cette thèse et en particulier lors de la rédaction ^_^, elle m'a permis de garder le moral ! Je tiens aussi à remercier mes parents qui m'ont permis d'étudier à l'université et d'atteindre la thèse. Je tiens à souligner ici que me permettre de suivre de telles études n'avait rien d'évident : d'abord parce que dès avant l'université il devient difficille d'expliquer exactement ce qu'on apprend à l'école et à quoi ça sert vraiment. Ensuite parce que en cette période de chômage les diplômes ne sont pas une garantit absolue de trouver un travail, or la préoccupation principale et bien légitime de mes parents était tout de même de me faire suivre une formation qui débouche sur un « bon » travail. Ca m'a valu les propositions de travailler dans le bâtiment (en 3ième), l'armée (pendant le lycée) ou la SNCF (après le BAC) avec toujours pour motivation de trouver un travail « sur ». Mais j'ai toujours pu décider de ce que je faisais, c'est-à-dire que mes parents ont bien voulu me faire confiance et payer des études dont ils ne savaient rien ou presque et dont je ne savais pas moi-même vraiment où elle pouvaient me mener…mais qui m'ont finalement mené ici. Donc encore une fois merci à eux ☺

Je tiens ensuite à remercier l'équipe IIHM dans son ensemble, j'y ais évolué (dans tous les sens du terme) pendant 7 ans au contact de ses divers membres et j'ai pu y mesurer l'importance du terme « équipe », de l'ambiance qui peut y régner et de l'impacte que cela peut avoir sur le travail et le bien être de chacun. Je remercie aussi toute l'équipe et en particulier ceux du bowling pour le cadeau de départ qui m'a été très utile par la suite ☺. Je remercie aussi tout spéciallement Gaelle Calvary qui a été mon encadrante pendant tout ce temps. En particulier je la remercie de sa bonne humeur quasi perpétuelle qui a rendu le travail commun très agréable☺. Elle a su cadrer mon travail tout en me laissant une grande liberté. C'est en grande partie grace à elle que mes recherches ont pu progresser et je garderais toujours en référence ces réunions qui nous permettez de progresser dans notre compréhension des problèmes et solutions, que ce soit au labo, à l'hotel, dans un train, un couloir ou un hopital …ce qui m'amène aussi de la remercier, ce qui est moins courant sans doute, de m'avoir sauver la vie lors d'une mission mal engagé à Bucarest car sans son soucis à mon égard, soucis qui dépasse ce que l'on peut attendre d'une encadrante de thèse, j'aurais pu trépasser une longue nuit à l'hotel ^_^. Je remercie aussi Jean et Théodora qui m'ont eux aussi sauvé la vie en m'emmenant à temps à l'hopital et en prenant ensuite le temps de traduire mon dossier médical pour me permettre d'être rapatrié saint et sauf à Grenoble. Toujours dans l'équipe IHM, je tiens à remercier Joelle Coutaz et d'abord car c'est elle qui donne à l'équipe les moyens de s'épanouir. J'ai aussi pu mesurer lors des réunions de travail sa vision de l'IHM, une vision intégratrice, qui vient de loin et qui porte loin. Nous nous sommes parfois engeulé sur des sujets politiques ou autres, mais cela m'a permis d'apprécier aussi sa capacité à avoir des discussions franches et à faire la part des

choses, jamais ces engueulades n’ont eut de répercution sur nos relations de travail ni amicales. Je remercie aussi Nicole, Valérie et Annie qui formèrent le trio administratif de choc pendant longtemps sur lesquelles nous avons toujours pu compter et sans lesquelles la vie aurait été plus difficille car faite de plus de papiers incompréhensibles ^_^.

Enfin je tiens à remercier Philippe Palanque, Catherine Berrut, Eric Lecolinet, Jean-Claude Tarby et Jean Vanderdonckt les membres du jury de m’avoir fait l’honneur d’être mon jury de thèse.

Table des matières

INTRODUCTION

Cette thèse se situe dans le domaine de l'ingénierie de l'Interaction Homme-Machine. Elle vise à fournir des méthodes, modèles et outils pour la *plasticité* des Interfaces Homme-Machine (IHM). La propriété de plasticité a été définie par [98 Thévenin 1999] [99 Thévenin 2001] comme étant « la capacité d'une IHM à s'adapter à son contexte d'usage dans le respect de son utilisabilité ». Cette propriété était à l'époque motivée par la diversité croissante des contextes d'usage. Un contexte d'usage est défini par un triplet <utilisateur, plate-forme, environnement>. La diversité concerne les trois volets du contexte d'usage :

- Utilisateurs : La diminution du coût des plates-formes a permis à un large public de s'en équiper. S'il existe encore des logiciels conçus pour un profil particulier d'utilisateurs (ex : les contrôleurs aérien), de nombreux logiciels s'adressent à une large population. C'est le cas typiquement des applications Internet, des logiciels de montage vidéo, de retouches d'images, de traitement de texte, etc. Ces logiciels doivent, en conséquence, proposer des fonctions et des IHM adaptées à l'utilisateur voire personnalisables (ex : les portails individuels comme http://www.netvibes.com).
- Plates-formes : Les plates-formes se sont considérablement diversifiées depuis une vingtaine d'années. Avec les progrès en miniaturisation, la gamme des appareils incorporant des ordinateurs a explosé : téléphones portables, PDA, consoles, télévisions, GPS, autoradios, ordinateurs de bord, électroménager, chaînes HIFI, etc. Parallèlement, les dispositifs d'entrée/sortie se sont aussi diversifiés. Le clavier, la souris et l'écran n'ont pas disparu mais côtoient désormais écrans tactiles, vidéo projecteurs, synthétiseurs vocaux, manettes équipées d'accéléromètres, caméras permettant de faire du suivi de doigts, etc.
- Environnements : La diversité des plates-formes, en particulier des plates-formes mobiles, fait, en corollaire, varier les environnements. L'utilisateur n'est plus confiné à un lieu fixe (bureau, maison, etc.). Il interagit désormais dans la rue, les transports, etc.

Les premiers travaux en plasticité se sont concentrés sur la diversité des contextes d'usage. Plus récemment, les caractères variable et imprévisible du contexte d'usage ont été intégrés à la réflexion :

- Variable : Pendant l'interaction, l'état de l'utilisateur peut changer (ex : fatigue, expérience, etc.). Des plates-formes peuvent arriver, d'autres disparaître. Enfin l'environnement peut également changer (ex : changement de luminosité).
- Imprévisible : Les contextes d'usage ne sont pas tous prévisibles à la conception. Typiquement, il est utopique, sur le long terme, de penser avoir imaginé toutes les plates-formes et combinaisons de plates-formes possibles

à la conception. Contraindre un système interactif aux seuls contextes d'usage prévus à la conception, c'est peut-être priver l'utilisateur d'usages séduisants. De même, la dynamicité du contexte d'usage n'est pas plus prévisible.

Des propositions sont à l'étude pour la variabilité des contextes d'usage [50 Ganneau 2007] [45 Florins 2004] [46 Florins 2006]. En revanche, l'imprévisibilité est un problème ouvert, en particulier non traité par les outils de construction d'IHM plastiques. Ces manques motivent mon sujet.

I. Sujet

L'adaptation des logiciels est un problème complexe qui mobilise différentes communautés : Génie Logiciel, Système, IHM, etc. Par exemple, en Génie Logiciel, des travaux sont menés sur l' « Autonomic computing » [54 Horn 2001] [55 Kephart 2003]. En Système, des travaux sont menés sur l'adaptation des applications à la dynamicité des ressources. Ces travaux donnent lieu à des intergiciels généraux [2 Alvin 2003] [21 Capra 2003]. J'adopte ici le point de vue de l'Interaction Homme-Machine. Il s'agit de doter le concepteur d'IHM d'outils de construction d'IHM plastiques aptes à couvrir un contexte d'usage varié, variable et imprévisible.

En IHM, j'identifie trois gammes de travaux qui apportent des éléments de réponse à la plasticité : l'Ingénierie Dirigée par les Modèles (IDM) ; les boîtes à outils (BàO) d'interacteurs plastiques et enfin les approches de type Operating System (OS) touchant au système d'exploitation des plates-formes cibles. La Figure I-1 situe ces approches selon le niveau d'abstraction du système interactif qu'elles manipulent. Les approches IDM ont produit toute une gamme de modèles de niveaux d'abstraction divers allant des tâches utilisateur aux interacteurs. Dans une approche descendante, le code est généré par transformation de ces modèles : la transformation est dite de type Concrétisation. Inversement, les modèles peuvent être découverts à partir du code : la transformation est alors dite de type Abstraction. En intermédiaire, dans les approches de type BàO, l'IHM est construite par assemblage d'interacteurs. Enfin, les approches de type OS agissent au niveau du système d'exploitation (par exemple, le système de fenêtrage), produisant en conséquence une plasticité non limitée à un seul système interactif.

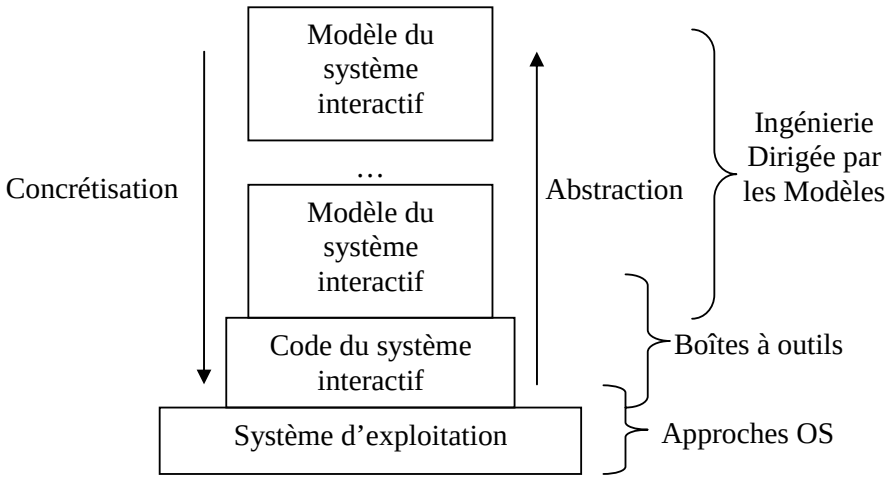


Figure I-1 : Positionnement des trois approches clé en plasticité des IHM.

Historiquement, les premiers travaux en plasticité des IHM se situent dans le champ de l'IDM. Une certaine maturité est aujourd'hui acquise. En particulier, un consensus est établi sur les niveaux de modélisation des IHM. Le cadre de référence [16 Calvary 2003] du projet européen [22 CAMELEON] résume ces acquis (Figure I-2).

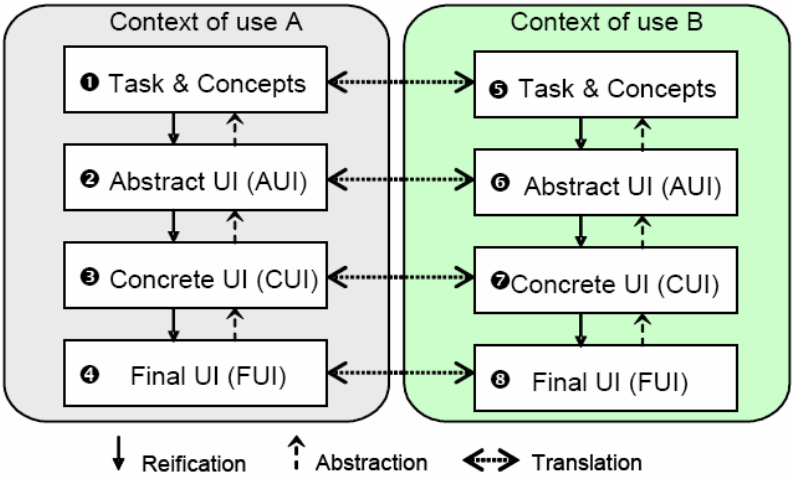


Figure I-2 : Cadre de référence CAMELEON.

Le cadre de référence distingue quatre niveaux d'abstraction. Le niveau Tâches et Concepts (1 et 5 sur la Figure I-2) décrit la sémantique de l'application du point de vue de l'utilisateur. La description doit normalement être indépendante de toute modalité. Le niveau Interface Abstraite (AUI) (2 et 6 sur la Figure I-2) modélise le dialogue entre l'utilisateur et le système. A ce niveau, des dépendances à la modalité peuvent être introduites. Typiquement, le dialogue peut varier du graphique au vocal. Le niveau Interface Concrète CUI (3 et 7 sur la Figure I-2) modélise l'IHM en termes d'interacteurs concrets (bouton, liste, etc.). Enfin, le niveau Interface Finale (FUI) (4 et 8 sur la Figure I-2) modélise l'IHM dans une technologie particulière (JAVA-SWING, PHP-HTML, etc.).

Prenons l'exemple de Sedan-Bouillon [7 Balme 2004], un site WEB plastique que nous avons développé dans le cadre du projet européen [22 CAMELEON]. Ce site permet à un usager de découvrir la région de Sedan et Bouillon. Il peut, s'il le souhaite, y réserver une chambre d'hôtel. Deux IHM « sur mesure » ont été créées : une pour PC, l'autre pour PDA. Ces IHM sont structurées en trois espaces de dialogue (Figure I-3) : un titre (en haut sur PC ; non affiché sur PDA), une barre de navigation (à gauche sur PC ; en haut sur PDA) et un contenu (au centre sur PC et PDA). Les IHM diffèrent par l'agencement spatial et les choix de présentation.

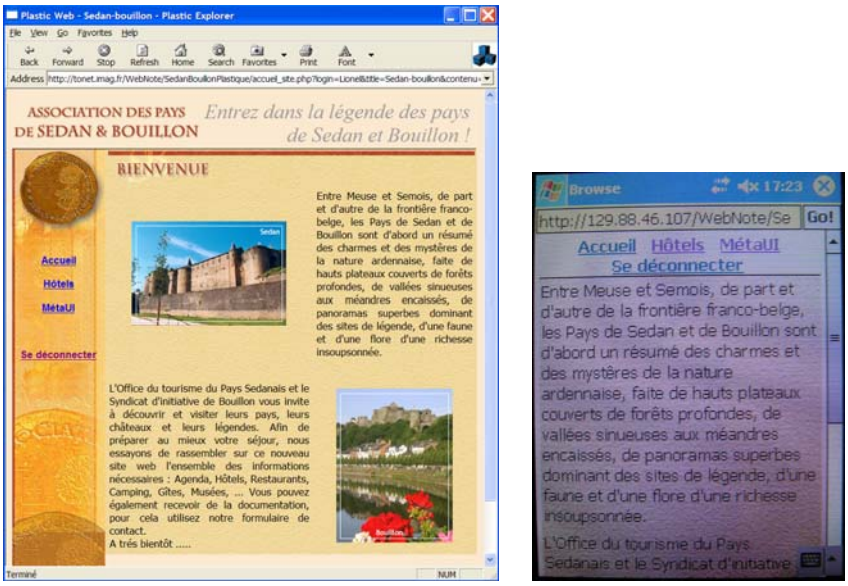


Figure I-3 : Version plastique du site de Sedan-Bouillon : à gauche, l'IHM PC ; à droite, l'IHM PDA.

Supposons qu'un utilisateur du nom de Lionel se connecte au site plastique de Sedan-Bouillon via un PC puis un PDA. L'arrivée du PDA est vue comme une opportunité pour étaler l'IHM entre les deux plates-formes. Une proposition de redistribution est faite (Figure I-4) : le site est structuré en un titre, une barre de navigation et un contenu. Lionel peut afficher là où il le souhaite les différents espaces de dialogue. Lionel choisit d'avoir le titre et le contenu sur PC (log_Lionel_0 sur la Figure I-4) et de disposer du titre et de la barre de navigation sur PDA (log_Lionel_1 sur la Figure I-4). La redistribution s'opère alors (Figure I-5). On notera que, dans ce prototype, l'adaptation est placée sous le contrôle de l'utilisateur (Figure I-4). Ce contrôle explicite requiert une IHM que nous appelons Méta-IHM [28 Coutaz 2006] [88 Roudaut 2006]. La méta-IHM rend observable voire contrôlable à l'utilisateur le processus d'adaptation.

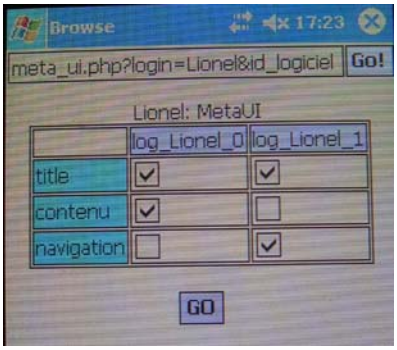


Figure I-4 : Méta-IHM pour le contrôle de la distribution de l'IHM de Sedan-Bouillon.

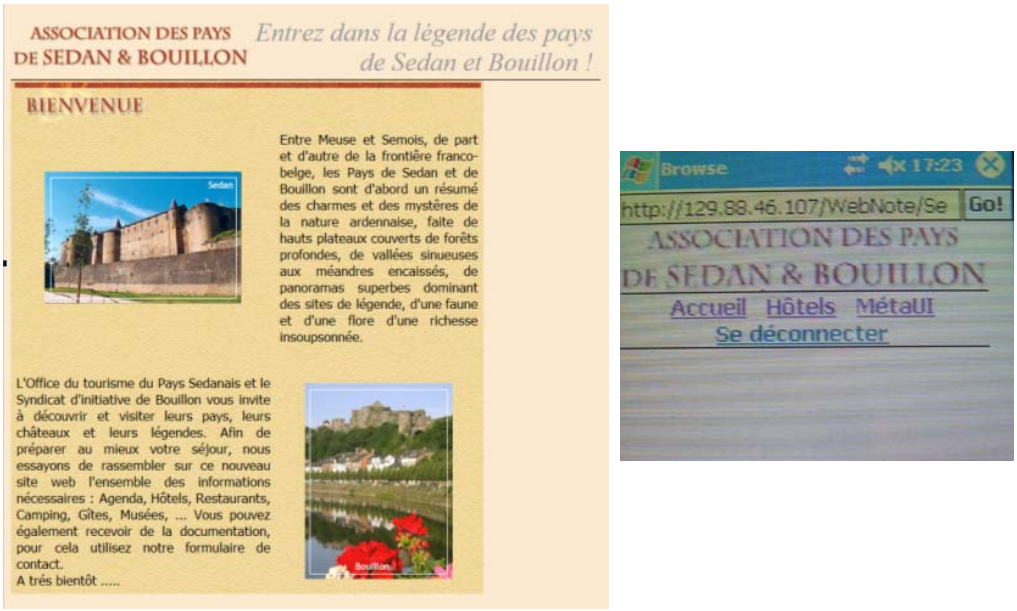


Figure I-5 : Version distribuée de Sedan-Bouillon : titre et contenu sur PC ; titre et navigation sur PDA.

Alors que, dans le prototype actuel de Sedan-Bouillon, toutes les IHM ont été préfabriquées (et codées), nous imaginons ci-après des extensions (non implémentées) illustrant, de façon plus complète, les besoins de plasticité.

Scénario n°1 : La grand-mère de Lionel arrive et décide de prendre part au choix de l'hôtel. Lionel lui confie le PDA. Malheureusement, ses doigts peu agiles et tremblants ne lui permettent pas d'interagir confortablement au stylet. Le site n'ayant pas été conçu pour personnes âgées, le système applique une *transformation* à l'IHM : les liens hypertextes de la barre de navigation sont transformés en larges boutons se répartissant la surface d'affichage (Figure I-6).

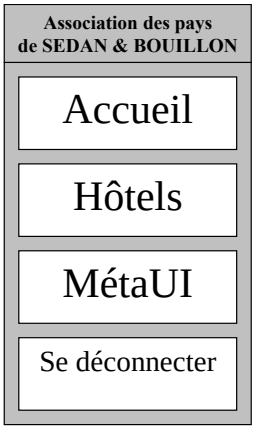


Figure I-6 : Maquette de l'IHM à gros boutons sur PDA.

Scénario n°2 : Alex, qui finissait la vaisselle, arrive. Il décide de découvrir lui aussi la région de Sedan-Bouillon. Il demande à Lionel s'il peut lui emprunter le PDA. Lionel, poli mais soucieux de voir les mains moites d'Alex, lui suggère d'utiliser son nouveau téléphone portable. Le téléphone d'Alex dispose, en effet, d'un module de reconnaissance vocale et peut se connecter au réseau domestique via Bluetooth. Alex, enthousiaste, allume son téléphone. Il décide d'utiliser l'oreillette et le microphone fournis de base. Les touches étant assez épaisses, il peut par moment piloter la navigation au clavier.

L'exemple enrichi de Sedan-Bouillon illustre bien les problèmes qui se posent aujourd'hui en plasticité des IHM :

- L'adaptation peut agir à tout niveau d'abstraction (Concepts et tâches, AUI, CUI, FUI) : on parle de remodelage. L'adaptation peut modifier la distribution de l'IHM sur l'ensemble des plates-formes : on parle de redistribution. Dans les deux cas, il faut être capable de décrire et transformer l'IHM. L'adaptation peut être automatique ou placée sous le contrôle de l'utilisateur via une Méta-IHM.
- Plusieurs technologies de rendu d'IHM existent (WEB, WAP, SWING, OpenGL, Vocal, etc.). Ces technologies sont, la plupart du temps, incompatibles entre elles et parfois non disponibles sur certaines plates-formes. Par exemple, rares sont les téléphones permettant d'afficher de l'OpenGL. Une IHM plastique doit pouvoir s'adapter à la technologie de rendu.
- Le système interactif doit pouvoir s'enrichir dynamiquement de nouvelles IHM. Ces IHM peuvent être conçues « sur mesure » pour un contexte d'usage particulier.
- Les différentes versions d'IHM doivent être cohérentes entre elles pour le confort de l'utilisateur.

Alors que de nombreux efforts ont jusqu'ici été portés sur les langages de description d'IHM (approche IDM), mon sujet porte sur les BàO d'interacteurs. J'énonce mes objectifs dans la section suivante.

II. Objectifs

Mes objectifs sont :

1. De couvrir le remodelage à tout niveau d'abstraction, allant du niveau Tâches-Concepts au code. La redistribution ne fait pas partie de mes objectifs.
2. D'ouvrir les interacteurs sur plusieurs technologies, leur permettre de commuter de l'une à l'autre et de les combiner. Par exemple, à un instant donné, pouvoir afficher un interacteur à la fois en HTML et SWING.
3. D'ouvrir les interacteurs sur des présentations non forcément prévues à la conception mais exploitées de façon opportuniste. Par exemple, pouvoir profiter d'une IHM à larges boutons subitement mise à disposition pour personnes âgées.
4. De prendre en compte des présentations « sur mesure ». Par exemple, pouvoir incorporer une barre de navigation « sur mesure » pour téléphones Bluetooth.
5. De placer l'interacteur sous le contrôle de l'utilisateur.

De ces objectifs découle une grille de requis (Tableau I-1). Cette grille me servira de grille d'analyse de l'état de l'art.

Tableau I-1 : Grille de requis.

Requis	Description du requis

Niveau sémantique = Tous	Le niveau sémantique indique, dans le cadre de référence CAMELEON, si la sémantique de l'interacteur est proche de la tâche utilisateur, de l'interface abstraite, concrète ou finale. Cette considération correspond à l'objectif n°1.
Empreinte technologique = Multi et non exclusives	L'empreinte technologique cerne les technologies de mise en œuvre possibles pour l'interacteur. Si plusieurs technologies sont disponibles (Multi), il faut examiner si elles le sont simultanément ou pas. Cette considération correspond à l'objectif n°2.
Extensibilité = Oui et dynamique	L'extensibilité examine la capacité de l'interacteur à être enrichi de capacités non prévues à la conception. Elle examine dans quelle mesure et dans quelles conditions cette extension peut être faite. En particulier, des capacités « sur mesure » peuvent-elles être ajoutées ? Peuvent-elles l'être dynamiquement ? Cette considération répond aux objectifs n°3 et 4.
Malléabilité = Totale	La malléabilité examine la capacité de l'interacteur à être transformé à la conception et à l'exécution. Elle examine la puissance des transformations de remodelage applicables. Cette considération correspond à l'objectif n°1.
Contrôlabilité / Prévisibilité du rendu	La contrôlabilité examine le degré de contrôle accordé à l'utilisateur (concepteur et/ou utilisateur final) dans la manipulation de l'interacteur. A défaut d'une contrôlabilité totale, le rendu est-il prévisible ? Cette considération correspond à l'objectif n°5.

Pour atteindre ces objectifs, mon approche consiste à garder trace dans l'interacteur des raisonnements tenus à la conception.

III. Approche

Mon approche s'inscrit dans le domaine du génie logiciel pour l'IHM. J'explore la voie d'interacteurs :

- Décrits à tout niveau d'abstraction de façon statique et dynamique. Par dynamique, j'entends la description de l'interacteur déployé dans son système interactif et son contexte d'usage. Ces descriptions statique et dynamique serviront de base à l'adaptation. Par exemple, dans Sedan-Bouillon (Figure I-5), si on considère la barre de navigation comme un interacteur, sa description fera apparaître qu'il est rendu sur une plate-forme de type PDA ou PC selon le cas. Si la plate-forme change, le rendu sera remodelé (Figure I-3).
- Polymorphes [33 Daassi 2004], c'est-à-dire embarquant plusieurs présentations. Les présentations sont décrites à tout niveau d'abstraction.
- Extensibles, potentiellement à tout niveau d'abstraction.

Le polymorphisme et l'extensibilité s'entendent aussi dynamiquement. Aussi, mon état de l'art couvre-t-il les langages de transformation et les Approches Orientées Services (SOA). L'état de l'art est la première étape de ma démarche.

IV. Démarche et plan

Ma démarche se structure en trois temps. Elle comporte premièrement un état de l'art. Cet état de l'art s'organise en trois pôles :

- Une étude des approches IDM et OS par un de leur représentant : respectivement UsiXML et Façade. Cette étude vise à fournir une compréhension des possibilités offertes par ces approches. J'utiliserai la grille du Tableau I-1 comme grille d'analyse de ces travaux.
- Une étude détaillée des boîtes à outils d'interacteurs permettant de prendre en charge certains aspects de la plasticité. Cette étude vise à identifier les acquis et manques de ces BàO. Là encore, j'utiliserai la grille du Tableau I-1 comme grille d'analyse de ces travaux.
- Enfin, une étude de domaines connexes mais utiles dans le cadre de ce travail : les langages de transformation et les annuaires dans le cadre des approches orientées services. Cette étude vise à fournir une compréhension des possibilités offertes par ces approches dans le cadre des IHM plastiques.

Dans un deuxième temps, je présente mes contributions :

- Un modèle d'Ecosystème décrivant le système interactif déployé dans son contexte d'usage. Ce modèle intègre des descriptions à tout niveau d'abstraction et permet de raisonner sur le système interactif « en contexte ».
- Les COMET : Une boîte à outils d'interacteurs de niveau sémantique la tâche utilisateur. Les présentations de ces interacteurs sont extensibles et utilisent potentiellement des technologies de rendus différentes.
- Le GDD : Un réseau sémantique jouant le rôle d'annuaire de systèmes interactifs.
- CSS++ : Un langage de style dédié à la plasticité des IHM.

Je décris enfin un démonstrateur CamNote++ mettant en scène ces contributions, à l'exception de l'Ecosystème non implémenté à ce jour. Le plan du rapport suit cette démarche. Il s'articule en trois volets : état de l'art, contributions et démonstrateur.

ETAT DE L'ART

Ce chapitre se structure en trois parties. Dans un premier temps, j'examine les approches IDM et OS à travers un de leurs représentants : UsiXML (http:/www.usixml.org) pour l'IDM ; Façade pour les approches de type OS. Cet examen me permet de comprendre les possibilités offertes par ces approches et de situer comparativement les BàO. La deuxième section contient un état de l'art des BàO d'interacteurs utiles à la plasticité. Enfin, dans la troisième section, j'étudie deux domaines pouvant servir d'outils à la plasticité : les langages de transformation et les annuaires dans le cadre des approches orientées services. Le chapitre se termine par une conclusion qui fixe les orientations de mes contributions.

I. Approches IDM et OS pour la plasticité

Dans une première section, j'examine UsiXML comme représentant des approches IDM. Fondamentalement, UsiXML permet de décrire une IHM à différents niveaux d'abstraction. Ces descriptions sont reliées entre elles par le biais de « mappings ». Les transformations ne se limitent pas à des concrétisations. Un arsenal d'outils UsiXML est disponible parcourant les différents chemins [103 Vanderdonckt 2004] du cadre de référence CAMELEON [16 Calvary 2003]. UsiXML n'est pas la seule tentative, loin de là. Mais sa couverture en termes de modèles, en particulier l'existence du Mapping modèle et la richesse des outils en font un représentant certain.

La seconde section examine les approches OS par leur représentant Façade (http://insitu.lri.fr/metisse/facades/). Fondamentalement, il s'agit de proposer des méthodes permettant d'adapter le système interactif a posteriori, sans que cette plasticité n'ait été prévue à la conception. L'adaptation intervient à l'initiative de l'OS. Elle peut porter sur le code de l'application ou sur la plate-forme logicielle d'exécution (OS, gestionnaire de fenêtre, etc.).

I.1. UsiXML : illustration des approches IDM

UsiXML répond aux besoins suivants [103 Vanderdonckt 2004] [69-70-71 Limbourg 2004] :

- Expressivité des descriptions d'IHM : Une IHM est décrite, selon différentes perspectives, par un ensemble de modèles éditables, calculables et manipulables par le système.
- Centralisation des descriptions : Chaque modèle est stocké dans une base de modèles contenant des modèles d'IHM exprimés dans un même langage.
- Approche transformationnelle : Chaque modèle de la base peut être sujet à transformations. Ces transformations couvrent les étapes de développement d'IHM ;

- Chemins de développement multiples : Les étapes de développement peuvent être combinées pour former des chemins de développement. Ces chemins doivent être compatibles avec les contraintes d'organisation, les conventions et le contexte d'usage. Par exemple, une série de transformations peut dériver une FUI à partir d'un modèle de tâches.

UsiXML répond à ces besoins en définissant un ensemble de modèles décrivant l'IHM selon différentes perspectives (Expressivité). Les quatre niveaux d'abstraction du cadre de référence CAMELEON sont couverts. La Figure II-1 illustre ces quatre niveaux sur le cas de l'activateur (le bouton). Dans une démarche descendante, la tâche d'activation peut être réifiée en un Objet d'Interaction Abstrait (AIO) de type Contrôle. L'AIO peut, à son tour, être réifié en AIO de modalités différentes : logicielle, d'une part ; physique d'autre part. Les réifications s'appliquent pas à pas jusqu'à obtenir des boutons effectifs (SWING, 3D, physique, etc.). On note la possibilité de réifications multiples à un niveau d'abstraction donné. Par exemple, au niveau CUI, l'AIO logiciel est affiné en un bouton poussoir 2D ou 3D.

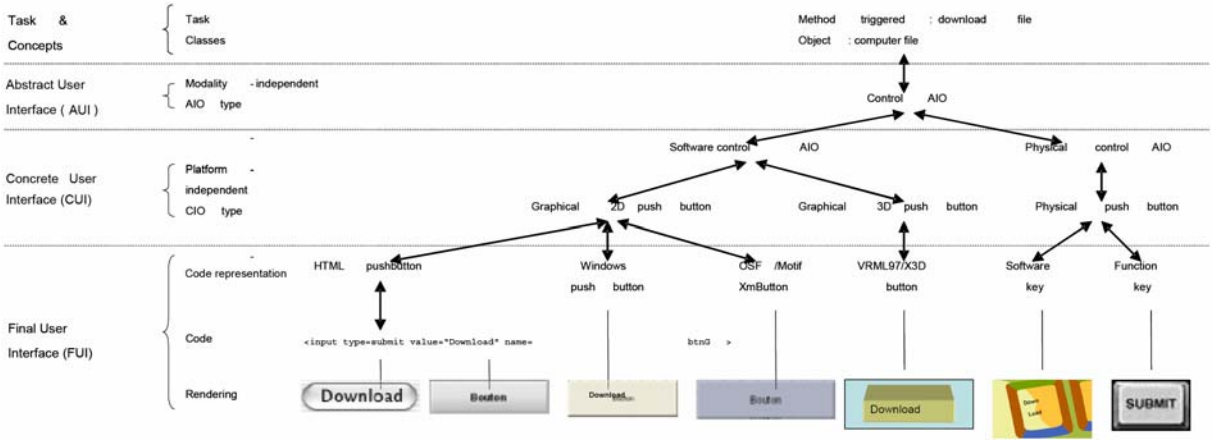


Figure II-1 : Illustration des niveaux d'abstraction de CAMELEON sur le cas de l'activateur. Extrait de [103 Vanderdonckt 2004].

En plus de ces quatre niveaux (Figure II-2), UsiXML intègre la modélisation du contexte d'usage. Il contient aussi un modèle de transformation s'appliquant à n'importe quel modèle UsiXML (Approche transformationnelle). Il intègre enfin un modèle de mapping permettant de mettre en relation des modèles de différents niveaux d'abstraction (Chemins de développement multiples). Tous ces modèles sont exprimés dans un même langage (Centralisation).

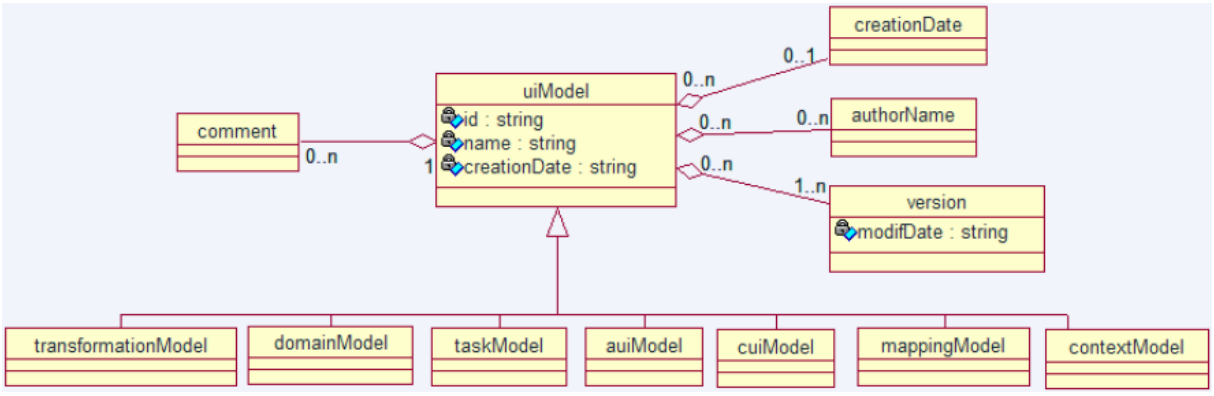


Figure II-2 : Les différents modèles d'IHM dans UsiXML.

En pratique, le modèle de domaine est équivalent aux diagrammes de classes UML. Le modèle des tâches est équivalent à CTT [84 Paterno 1997]. Ces modèles étant largement étudiés, je me focalise ici sur les langages d'AUI, de CUI et de mappings, bien moins étudiés.

I.1.A. Le langage d'AUI dans UsiXML

Le modèle d'AUI décrit, d'une part, les groupements entre les éléments de l'IHM, d'autre part, la navigation entre ces éléments. Le modèle d'AUI est peuplé d'AIO qui peuvent être de deux types : les *containers* et les *components*. Un container peut contenir plusieurs AIO. Un component est défini par plusieurs facettes modélisant ses capacités d'entrée/sortie, de navigation et de contrôle (connexion au Noyau Fonctionnel). Des relations autres que la composition peuvent être définies, telles que des relations spatiotemporelles (extrait de [1 Allen 1981]), des renforcements mutuels (mutual emphasis), des relations de proximité ou encore des relations de dialogue (basées sur les opérateurs LOTOS) (Figure II-3). Les renforcements mutuels entre plusieurs éléments d'IHM expriment que ces éléments doivent être différenciés d'une façon ou d'une autre, de sorte à ne pas pouvoir les confondre.

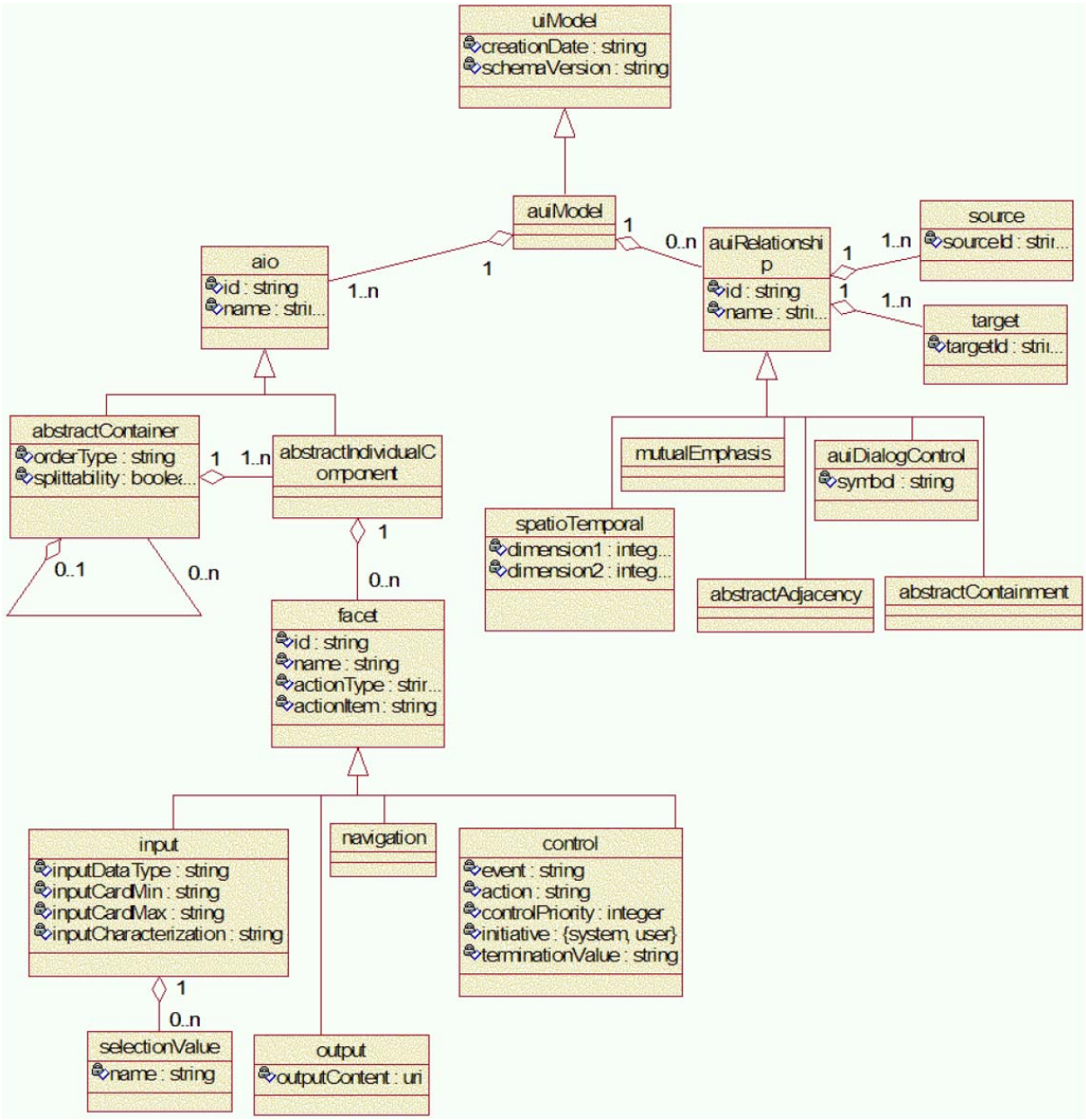


Figure II-3 : Diagramme de classe du langage AUI utilisé dans UsiXML.

La section suivante traite du langage de CUI.

I.1.B.Le langage de CUI dans UsiXML

Le modèle de CUI est trop complexe pour être ici présenté en détail. Les références peuvent être trouvées sur le site WEB de UsiXML. Notons toutefois que UsiXML définit deux sous langages de CUI : un pour les IHM graphiques ; l'autre pour les IHM vocales. Ces langages doivent pouvoir tenir le rôle de langages pivots, c'est-à-dire que des IHM définies par ce biais doivent ensuite pouvoir être rendues à l'aide de BàO existantes.

Le langage de CUI vocal est proche du pouvoir d'expression de VoiceXML. C'est un langage simple (peu de vocabulaire et de structure) visant à la reconnaissance de mots hors contexte. Le langage de CUI graphique couvre les IHM de type formulaire et, dans une certaine mesure, les IHM 3D. On y retrouve tous les widgets classiques d'intérêt

général (bouton, potentiomètre, arbre, menu déroulant, palettes de couleurs, de fontes, etc.).

La section suivante traite du langage de Mapping.

I.1.C. Le langage de Mapping dans UsiXML

Le langage de Mapping est la force de UsiXML et, de façon plus générale, de l'IDM. Il modélise les relations entre modèles et éléments de modèles (Tâches-Concepts, AUI, CUI, FUI, contexte d'usage, etc.). Le langage de Mapping permet de maintenir les liens entre les différentes représentations de l'IHM. Dès lors, une modification qui survient à un niveau quelconque peut être propagée aux autres niveaux grâce à ces mapping. La Figure II-4 décrit le métamodèle de mappings de UsiXML. Outre les abstractions et réifications classiques, des mappings spécifiques sont introduits. Il y a notamment la connexion des éléments d'IHM (AUI ou CUI) au noyau fonctionnel (trigger, observe, update) et la connexion des éléments d'IHM (C&T, AUI, CUI) au contexte d'usage (isShapedFor, isAllocatedTo, isDelegatedTo, hasContext). Le langage introduit aussi des transformations intra niveau d'abstraction (isTranslatedInto) ; des relations spécifiques entre modèles pour exprimer, par exemple, la manipulation des concepts dans les tâches (manipulates) ou la réalisation d'une tâche dans un AUI (isExecutedIn) (Figure II-4).

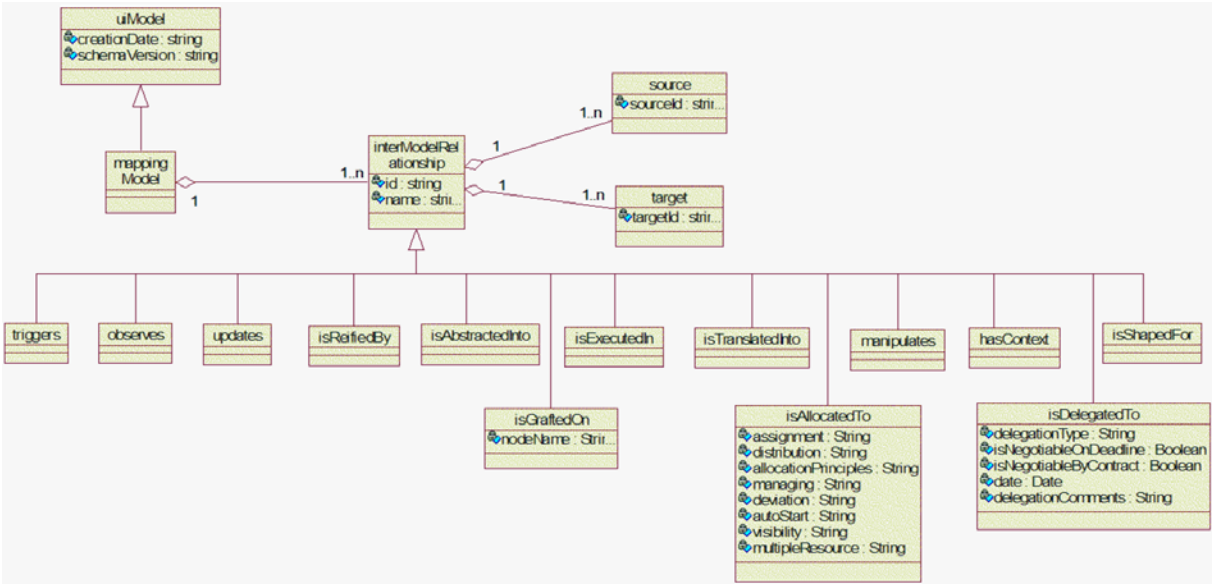


Figure II-4 : Métamodèle de Mapping dans UsiXML.

L'intérêt du modèle de Mapping a été démontré dans plusieurs travaux [51 Griffiths 1999]. En particulier, [42 Florins 2006] exploite les mapping pour les règles de dégradation élégante. Une règle, bien que longitudinale, s'ancre à un niveau d'abstraction donné : Tâches-Concepts, AUI, CUI, FUI. Par utilisation des mappings, la dégradation est propagée à tous les niveaux, assurant une cohérence de l'IHM. Sur la Figure II-5, l'IHM est initialement structurée en un seul espace de dialogue (a). Par manque de place, l'AUI est scindée en deux espaces de dialogue coïncidant avec l'opérateur d'entrelacement entre sous tâches (b). Les niveaux CUI et FUI sont mis à jour conformément.

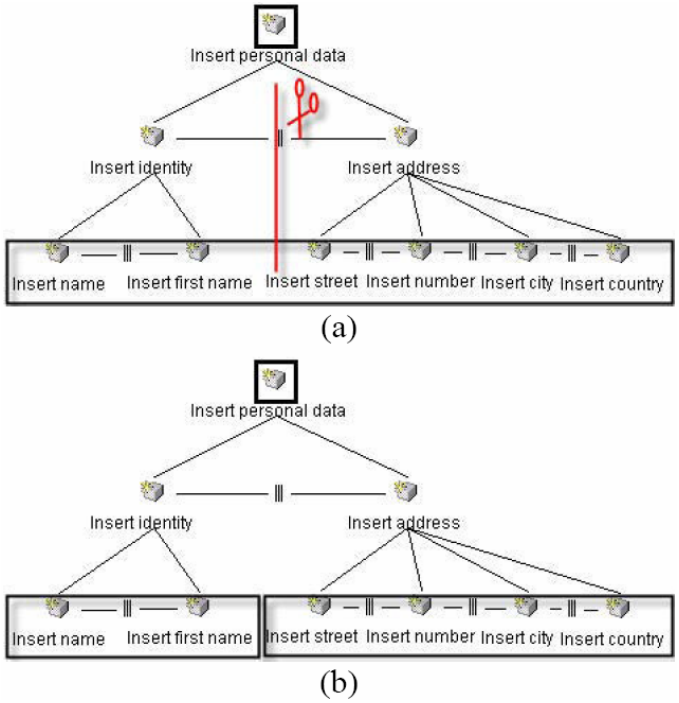


Figure II-5 : Exemple de dégradation élégante scindant l'espace de dialogue initial (a) en deux (b). La scission se fait au niveau de l'entrelacement des sous tâches

[67-68 Lepreux 2006] montre qu'il est possible de composer des IHM UsiXML au niveau CUI. Elle définit des opérateurs ensemblistes (union, union sans doublon, intersection, etc.) qu'elle applique à la description des deux IHM à composer. Pour l'union sans doublon, elle repère les éléments redondants au sein des IHM (par exemple, les deux champs textes du nom (Figure II-6). Aujourd'hui, l'identification est lexicale mais elle pourrait s'appuyer sur les mappings. Typiquement, les mappings permettraient de répondre à des questions telles que : « Ces deux champs texte sont-ils des incarnations d'une même tâche ? ». La considération des niveaux d'abstraction supérieurs (AUI et Tâches-Concepts) est une perspective au travail.

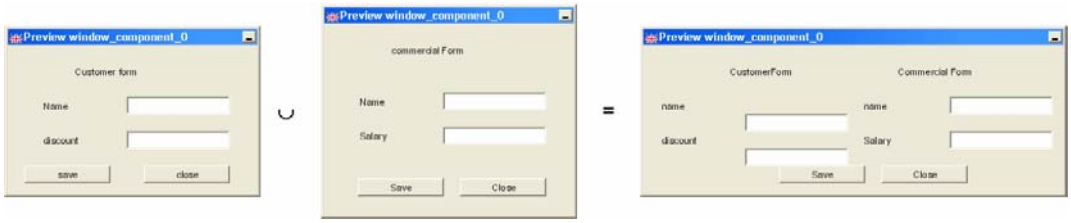


Figure II-6 : Union sans doublon de deux interfaces : les boutons Save et Close n'ont pas été dupliqués.

I.1.D. Conclusion

Comme de nombreux autres travaux (UIML, XAML, XUL, etc.), UsiXML redéfinit son propre langage pivot. Ce choix d'un langage pivot est pratique pour exprimer les mappings et transformations. En revanche, il limite du même coup les possibilités d'utilisation d'autres langages, sauf opérations de traduction (par exemple HTML ou XUL vers UsiXML). Mais la vraie limite que je vois au langage pivot est la difficulté à couvrir tous les types d'IHM. En particulier, les IHM de nouvelle génération (multi pointeurs, zoomable, rotation, 3D, entrées riches, etc.) ne sont pas bien prises en compte. Les auteurs d'UsiXML préviennent que ça n'est pas leur but. Ils annoncent explicitement que UsiXML n'est pas :

- un langage pour l'implémentation d'IHM. Il repose sur des interpréteurs ou des compilateurs qui traduisent les descriptions UsiXML en d'autre langages de spécification d'IHM (XHTML, Java-SWING, VRML…).
- une description bas niveau. En particulier les détails liés au système d'exploitation ne sont pas décrits.
- une infrastructure ou une machine virtuelle. Le rendu d'une interface UsiXML repose sur un interpréteur.
- un langage permettant de modéliser toutes les IHM ou tous les widgets. Il propose de n'en prendre en charge qu'une partie, la plus communément répandue.

De notre point de vue, la véritable force de UsiXML est d'avoir explicité les liens entre eux en tant que modèle à part entière : le modèle de Mapping.

UsiXML permet de décrire, au sein d'un même document, plusieurs solutions d'IHM selon les contextes d'usage. En cela, un document UsiXML est une forme de capitalisation de tous les modèles d'une IHM. Il contient en lui-même l'aptitude d'une IHM à couvrir différents contextes d'usages.

Les IHM « sur mesure » peuvent être partiellement prises en compte. Il existe bien des éléments CUI « sur mesure » pour certaines tâches (par exemple, la palette de couleurs - ColorPicker), mais il n'est pas possible d'en spécifier de nouveaux sans étendre le langage lui-même. Une telle extension est hors de portée du concepteur d'IHM. A plus gros grain, le concepteur peut cependant lier un modèle de CUI « sur mesure » à un modèle d'AUI ou de tâche. Mais ce modèle de CUI « sur mesure » ne peut être composé que d'interacteurs du langage de CUI.

En synthèse, le tableau ci-dessous analyse UsiXML au regard de nos objectifs.

Tableau II-1 : Grille d'analyse pour UsiXML.

Niveau sémantique	UsiXML est un langage permettant de décrire un système interactif à tout niveau d'abstraction, y compris son contexte d'usage. Des mappings peuvent être définis entre ces modèles.
Empreinte technologique	L'empreinte technologique des documents UsiXML dépend des interpréteurs disponibles. Notons à priori qu'un document ne peut pas être rendu en même temps selon différentes technologies, partiellement ou non.
Extensibilité	Le langage n'est pas extensible par les concepteurs d'IHM. Il n'est pas possible de rajouter de nouveaux interacteurs par exemple.
Malléabilité	Un modèle de transformation basé sur les grammaires de graphes est fourni. Hormis pour les interacteurs « sur mesure », ce langage offre la possibilité de spécifier des transformations diverses. L'étude détaillée des grammaires de graphes est faite en section III.1.C.
Contrôlabilité / Prévisibilité du rendu	La contrôlabilité et la prévisibilité du rendu dépendent du niveau de modélisation du document UsiXML. Si, par exemple, le document ne contient qu'un modèle de tâches, le rendu sera difficilement prévisible. En revanche, si la spécification va jusqu'au CUI, alors la prévisibilité est assurée. La contrôlabilité dépend du type de génération de l'IHM : de faible voire inexistant à total selon que la génération est automatique, semi-automatique ou manuelle.

I.2. Façade: illustration des approches « OS »

[96 Stuerzlinger 2006] met en œuvre une approche peu classique : il s'intéresse à l'adaptation des logiciels via le gestionnaire de fenêtres. En cela, on peut rapprocher Façade des travaux sur les Méta-IHM [88 Roudaut 2006].

Les auteurs constatent que la complexification fonctionnelle des logiciels entraîne une complexification des IHM alors même que la plupart des utilisateurs n'utilisent qu'une petite partie des fonctionnalités proposées. De cela, résultent des IHM parfois surchargées (ex : GIMP http://www.gimp.org) ou des IHM ayant quantité de menus (ex : Microsoft Word). Le problème auquel s'attèlent les auteurs est de permettre à l'utilisateur de réarranger son IHM pour un accès rapide aux fonctionnalités qu'il utilise vraiment.

L'approche des IHM adaptatives n'a pas été retenue par les auteurs. Les IHM adaptatives présentent, en effet, le défaut de surprendre les utilisateurs. S'appuyant sur les travaux de [44 Findlater 2004], les auteurs avancent que les utilisateurs ne sont pas plus performants à utiliser des IHM adaptables par rapport à des IHM statiques ou adaptatives. Par contre, il ressort de cette étude que les utilisateurs préfèrent les IHM adaptables aux deux autres.

L'approche retenue par les auteurs n'est pas de proposer un nouveau langage ou une nouvelle BàO, mais un nouveau gestionnaire de fenêtres. Les auteurs justifient ce choix par le fait que l'ajout de capacités d'adaptation aux BàO alourdirait leurs API déjà complexes et demanderait de nouveaux efforts aux programmeurs. De plus, dans une telle approche, ce serait au concepteur de prévoir ce qui peut être adapté, alors que les auteurs de Façade veulent que ce soit l'utilisateur qui soit totalement maître du processus d'adaptation.

Les auteurs formulent les requis suivants à propos des IHM adaptables :

- Adaptation rapide, simple et à la volée : L'utilisateur doit pouvoir adapter l'IHM à tout moment de façon simple, par exemple, avec des techniques de manipulation directe.
- Des adaptations locales, pas seulement globales : L'utilisateur devrait pouvoir spécifier des adaptations pour des sous parties de l'IHM. Ces adaptations ne doivent pas nécessairement être conservées entre les sessions. Par exemple, une barre d'outils peut être adaptée pour un document et une session particulière mais pas pour les autres documents.
- Adaptation profonde : Les adaptations possibles ne doivent pas être limitées à un petit ensemble d'options prédéfinies. L'utilisateur doit pouvoir copier, coller, couper, remplacer, dupliquer des éléments de l'IHM. Tout doit pouvoir être changé à l'écran, pas seulement la présentation mais aussi l'interaction.
- Adaptation inter-applications : Les adaptations doivent pouvoir se faire entre applications. Des éléments d'IHM d'une application doivent pouvoir être combinés à une autre.

Façade implémente la possibilité de copier n'importe quelle partie d'une IHM. La copie peut ensuite être placée dans une nouvelle fenêtre ou dans une IHM préexistante. Cela se fait simplement en détourant, sur l'IHM source, la partie à découper. Le détour se trace à la souris tout en appuyant sur une touche spéciale du clavier. La Figure II-7

illustre comment des copies d'éléments d'IHM de fenêtres différentes peuvent être regroupés : les éléments 1, 2 et 3 de la fenêtre de gauche ainsi que l'élément 4 de la fenêtre du milieu sont copiés dans la fenêtre de droite. Originaux et copies restent synchronisés. Les actions sur l'un sont répercutées sur les autres.

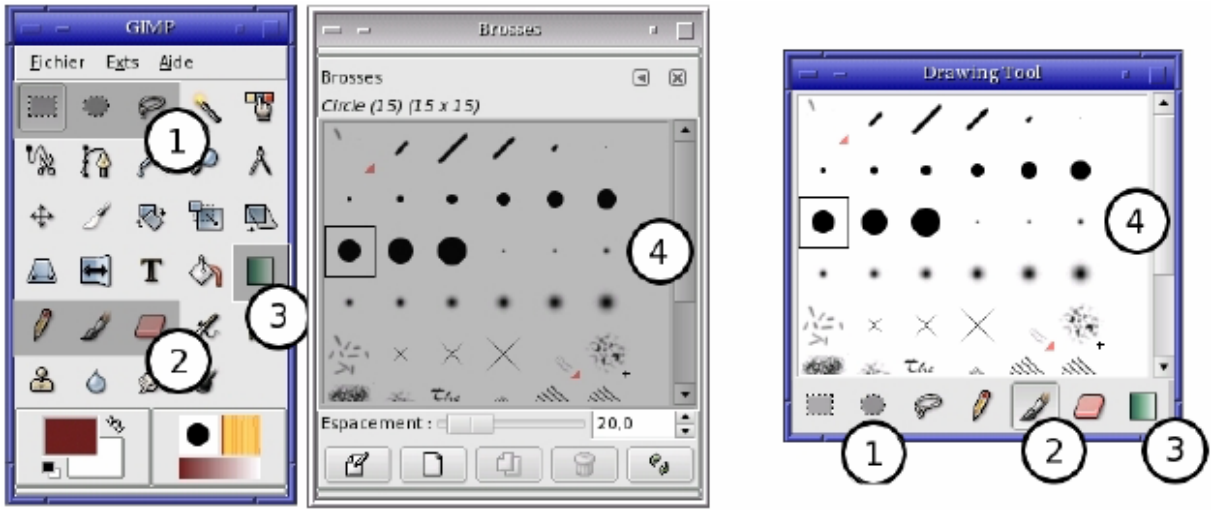


Figure II-7 : Façade : les éléments 1 à 4 des fenêtres de gauche et du milieu sont dupliqués et assemblés dans la fenêtre de droite.

Façade offre un moyen puissant de créer de nouvelles IHM à partir de morceaux choisis d'IHM existantes. Façade permet aussi de substituer à des éléments d'IHM d'autres présentations. C'est le cas, par exemple, d'un menu déroulant qui peut être substitué par une liste de boutons radio (Figure II-8) : l'utilisateur demande à remplacer le menu déroulant (fenêtre de gauche) par des boutons radio (fenêtre de droite). Via une Méta-IHM (fenêtre du milieu), l'utilisateur spécifie les options à conserver. Pour réaliser cela, Façade s'appuie sur les services d'accessibilité supportés par les BàO modernes (GTK, SWING, Qt, StarOffice, Mozilla…). Ces BàO permettent de récupérer des informations sur les widgets de façon standardisée (un standard différent pour Windows, un pour MAC OS, un pour X-Window). Encore une fois, cela se fait à l'insu de l'application. Celle-ci doit seulement pouvoir être interrogée par le gestionnaire de fenêtres via l'API dite d'accessibilité.

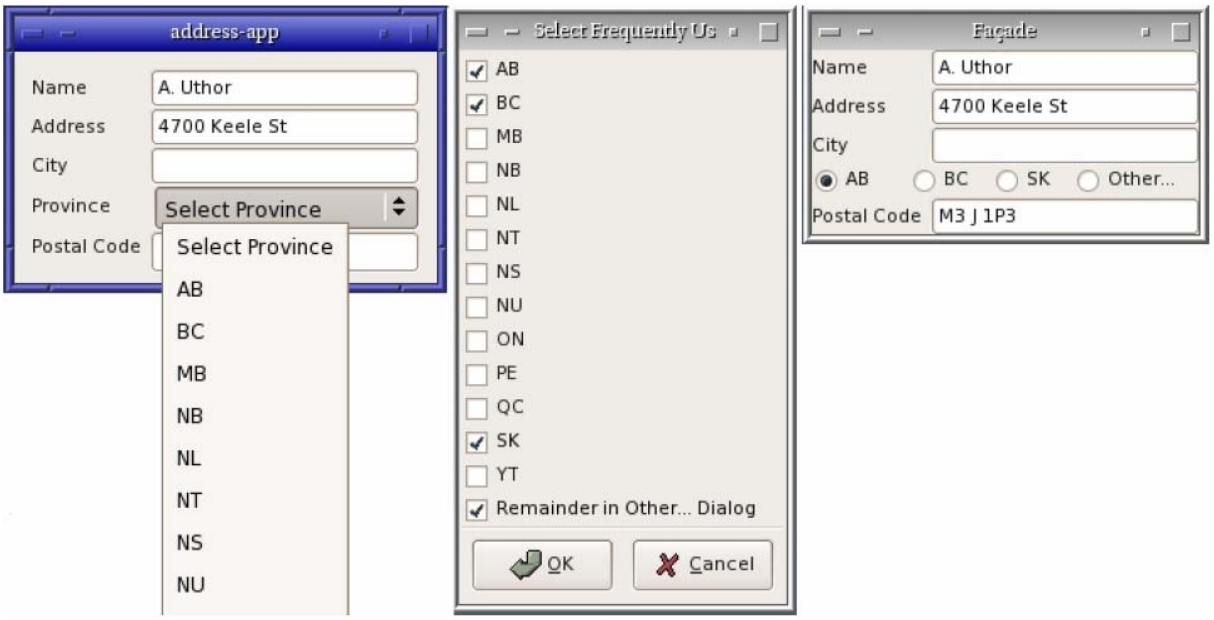


Figure II-8 : Façade : Substitution d'un menu déroulant (fenêtre de gauche) par des boutons radio (fenêtre de droite) sous le contrôle de l'utilisateur via une Méta-IHM (fenêtre du milieu).

La substitution d'un élément d'IHM par un autre n'est pas limitée aux éléments standardisés. Les auteurs montrent qu'il est aussi possible d'utiliser des IHM sur mesure : par exemple, une carte politique du Canada à la place d'un menu déroulant (Figure II-9). La seule contrainte est de respecter l'API de l'élément à remplacer.

Figure II-9 : Façade : des widgets sur mesure peuvent aussi être utilisés.

Enfin, les auteurs montrent que des techniques d'interaction comme l'orthozoom [4 Appert 2006] ou les toolglasses [13 Bier 1994] peuvent être implémentées au niveau du gestionnaire de fenêtres. Il suffit à l'utilisateur de sélectionner une fenêtre et d'appuyer sur une touche spéciale du clavier pour que la fenêtre devienne toolglass (Figure II-10).

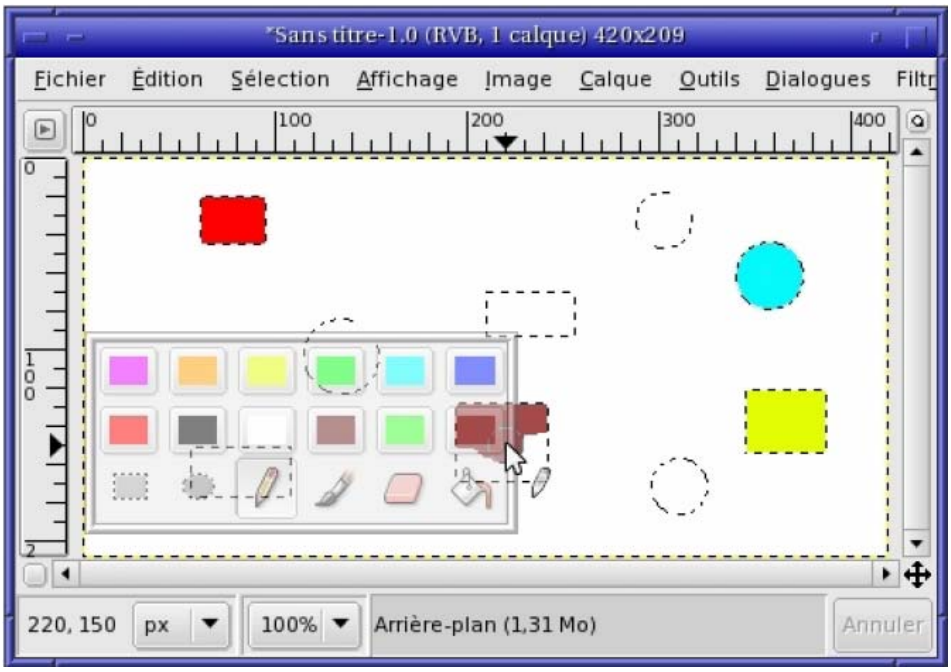


Figure II-10 : Façade : Les toolglasses sont couvertes par l'approche.

I.2.A. Conclusion

Façade propose une approche innovante, basée sur une redéfinition du gestionnaire de fenêtres. Ce travail montre la faisabilité d'une adaptation à ce niveau, sans donc que les applications soient prévues pour. Cela montre aussi la possibilité d'implémenter facilement, au niveau du gestionnaire de fenêtres, des techniques d'interaction évoluées comme les toolglasses ou l'orthozoom. Les applications en profitent donc sans avoir été prévues pour.

D'un point de vue de la plasticité, dans Façade, l'adaptation est placée sous le contrôle de l'utilisateur. L'utilisateur peut attacher/détacher des parties quelconques d'IHM, substituer une présentation par une autre, y compris avec des présentations sur mesure.

L'approche étant originale, il est intéressant d'essayer d'en cerner les limites du point de vue de la plasticité :

- Façade ne semble pas proposer d'adaptation d'ensemble. Par exemple, comment l'utilisateur pourrait il spécifier qu'il veut copier tous les boutons d'une IHM donnée ? Comment dire qu'il veut que tous les choix ayant plus de 10 éléments soient affichés sous la forme de menus ? La solution se trouve peut-être dans une méta IHM permettant de spécifier de telles requêtes qui pourrait ensuite être résolues en s'appuyant sur les services d'accessibilité. Les limites seront ici celles de la BàO ayant servi à la construction de l'IHM à adapter.
- Façade ne propose pas (encore ?) d'adaptation de style. L'utilisateur pourrait, par exemple, vouloir changer la fonte d'un texte ou la couleur de fond d'un panel. L'utilisation d'un langage de style à la CSS et/ou d'une Méta-IHM, s'appuyant sur les services d'accessibilité, devrait pouvoir régler le problème. Là encore, on retombe sur les capacités des BàO.

- Façade permet d'utiliser des interacteurs sur mesure, comme la carte du Canada. Mais comment savoir, dans le cas général, par quoi un widget peut être remplacé ?
- Comment les représentations des opérateurs entre tâches pourraient ils être pris en compte ? Par exemple, comment un entrelacement de tâches présenté sous forme de colonne (les espaces représentant les tâches étant mis les uns sous les autres) pourrait-il être transformé en onglets ?

Ces questions sont ouvertes. En synthèse, le tableau suivant caractérise Façade vis-à-vis de nos requis.

Tableau II-2 : Grille d'analyse pour Façade.

Niveau sémantique	Les objets manipulés sont du niveau pixel de l'interface finale (n'importe quelle zone peut être détourée). Il est aussi possible d'accéder à l'arbre des widgets (niveau FUI).
Empreinte technologique	Façade est un gestionnaire de fenêtres. Il n'exclut donc aucune technologie de développement d'IHM. En revanche, l'IHM est forcément graphique.
Extensibilité	Il semble possible d'ajouter des fonctions de substitution d'un interacteur par un autre. Mais la façon de procéder n'est pas clairement définie.
Malléabilité	L'utilisateur peut copier/coller/réassembler son IHM (niveau pixels ou FUI). Il peut aussi substituer des widgets par d'autres.
Contrôlabilité / Prévisibilité du rendu	L'utilisateur a un contrôle total sur les transformations.

I.3. Prise de recul

En synthèse, les approches « extrêmes » de type IDM ou OS (Figure I-1) présentent toutes deux des avantages certains pour la plasticité. En revanche, elles se heurtent à une même limite : celle des BàO sous-jacentes. La section suivante y est consacrée.

II. Boîtes à outils d'interacteurs plastiques

Cette section est dédiée à l'analyse des BàO existantes pour la construction d'IHM plastiques. Les BàO sont organisées selon leur centre d'intérêt principal. Je distingue les approches à dominante Polymorphisme, Multimodalité et Post-WIMP. Elles sont examinées dans cet ordre.

II.1. Dominante Polymorphisme

Le principe est celui d'interacteurs à rendus multiples. Les rendus sont principalement graphiques. Les BàO étudiées sont ACE, WAHID et XFORMS.

II.1.A. ACE

ACE est une approche à dominante Polymorphisme. L'objectif de ACE [59-60 Johnson 1992-1993] était de permettre la construction d'applications dans des domaines variés de la même façon qu'avec des tableurs (Speadsheets) [78 Nardi 1991]. ACE était motivée par un constat : les BàO de l'époque (ce constat reste globalement vrai) ne présentaient pas le niveau d'abstraction suffisant pour construire de telles applications. Ce niveau devrait être plus proche de la tâche utilisateur. ACE combine deux mondes : celui des BàO et des tableurs. Il offre plusieurs points d'entrées : à l'utilisateur final, au concepteur et au programmeur (Figure II-11.)

- Comme un tableur, ACE offre un ensemble d'éléments prédéfinis appelés frameworks (tableaux, graphes, cartes…). Ces éléments peuvent servir de point de départ au développement d'applications par des utilisateurs finaux ou intermédiaires.
- Comme dans une BàO, les frameworks proposés et plus généralement tous les composants utilisés sont implémentés en C++. Les programmeurs C++ peuvent donc les étendre.
- Comme dans un environnement de développement, le programmeur peut ajouter de nouveaux types de données, de nouveaux frameworks. Les composants peuvent être enrichis de nouvelles présentations. Les utilisateurs peuvent eux-mêmes choisir parmi les présentations possibles. Enfin, ACE offre un mini langage de programmation.

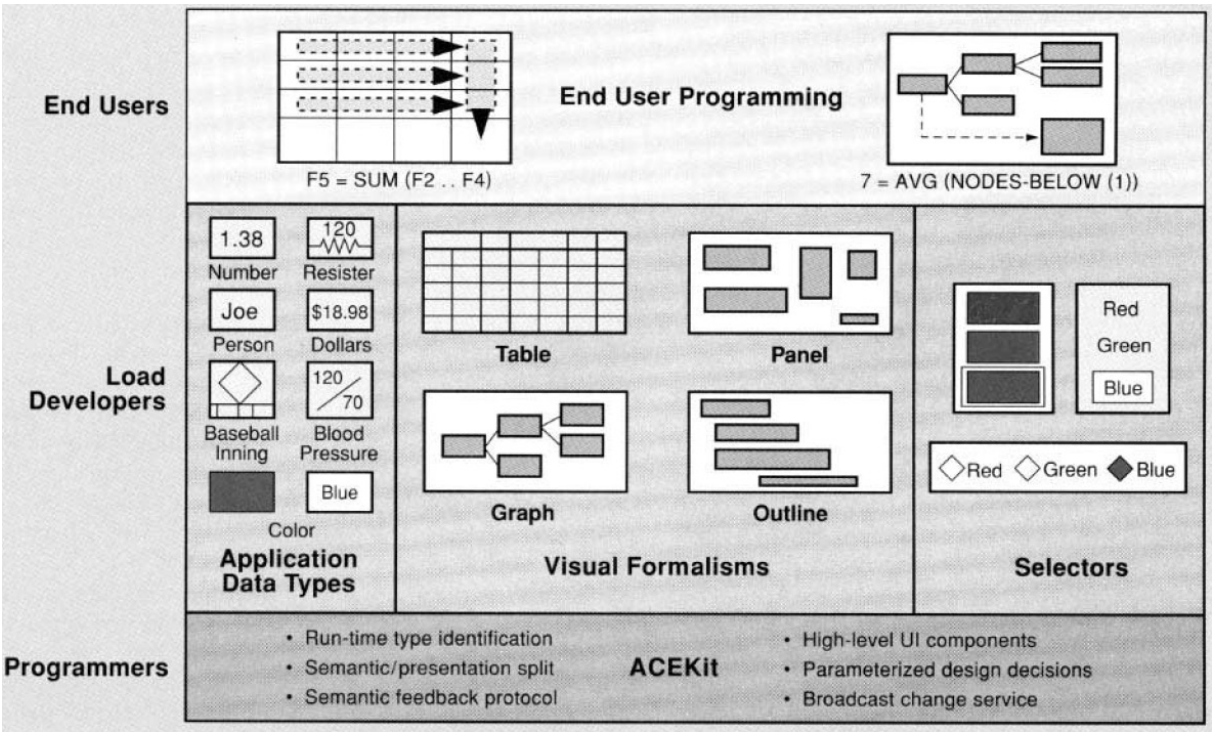


Figure II-11 : ACE offre des points d'entrées à différents types d'utilisateurs : utilisateurs finaux, concepteurs et programmeurs.

ACE se concentre sur le niveau tâche. Cela implique la possibilité de définir des tâches métier. Le problème est alors la prolifération des tâches et, en conséquence, la difficulté voire l'utopie de pouvoir fournir dans une BàO autant d'interacteurs qu'il y a de tâches métiers. ACE se concentre, en conséquence, sur les tâches et concepts d'utilité publique, c'est-à-dire largement répandus (ex. date, temps, fichiers, couleur…).

Les auteurs constatent que la plupart des widgets présents dans les IHM sont de deux types :

- Data Selectors : Ils représentent et permettent de manipuler des données de l'application (une liste de choix, un potentiomètre, etc.) ;
- Command Selectors : Ils représentent et permettent de déclencher des opérations de l'application (un bouton, un menu, etc.).

La classification se fait donc au niveau sémantique. J'examine ci après les Data Selectors car ils illustrent bien le principe de polymorphisme des présentations dans ACE.

Les Data Selectors

Les Data Selectors permettent à l'utilisateur de spécifier la valeur d'une donnée (Data) et de la visualiser. Il existe une infinité de types de données et de façons de les spécifier. Toutefois, les auteurs constatent qu'il est possible de les caractériser :

- Certains types sont discrets et de petite taille (ex : une bordure peut prendre les valeurs NONE, SOLID, DASHED, DOTTED). Un cas particulier souvent utilisé est le type booléen et ses dérivés (ex : TextBold : {ON, OFF}) ;
- D'autres types permettent la sélection multiple dans un ensemble discret (ex : Choix de plats dans un restaurant) ;

- D'autres types permettent la sélection d'une seule valeur dans un ensemble discret mais grand et doté de relations proches de celles caractérisant l'ensemble des entiers en mathématique (ex : Choix du volume entre 0 et 100) ;
- D'autres types enfin permettent la sélection de plusieurs valeurs ou sous ensembles dans un ensemble (ex : Choix d'heures ou de créneaux horaires).

Tous les types de choix peuvent être classés dans l'une de ces catégories. Il est ainsi possible de déterminer dans quelle mesure deux choix sont sémantiquement proches l'un de l'autre (choix simple ? Choix multiples ? Propriétés sur l'ensemble des choix ?).

La présentation du choix est prise en charge par les *Data Selectors Presenters*.

Les Data Selectors Presenters

Un *Data Selector Presenter* fixe la présentation du choix (pas celle des éléments à choisir – ces présentations sont prises en charges par les Data Presenters). Par exemple, un choix de couleurs peut se faire par une liste de boutons (un bouton par couleur) ou par un bouton cyclique (cycle-button) affichant une couleur à la fois et permettant de passer à la suivante de façon cyclique. Dans les deux cas, la couleur peut être représentée soit par son nom, soit par un rectangle coloré (Figure II-12). C'est le Data Selector qui est en charge de ce choix.

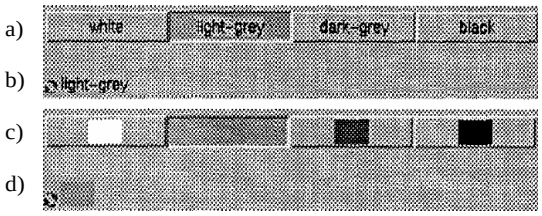


Figure II-12 : Quatre façons de choisir une couleur suivant que (1) le choix se fait dans une liste (a et c) ou par bouton cyclique (b et d) et (2) la couleur est représentée par son nom (a et b) ou un rectangle coloré (c et d).

Plusieurs « Presenters » peuvent être associés à un Data Selector. Les présentations sont synchonisées. Ainsi, tout changement dans une présentation est propagé aux autres présentations. La communication se fait via un protocole standardisé, indépendant du type de choix.

La combinaison des Data Selectors Presenters et des Data presenters fournit une large gamme de possibilités, bien plus large que dans les BàO classiques (SWING, TK..). L'avantage est aussi de raisonner en termes de tâches et non plus de widgets.

En résumé, ACE fournit :

- Un ensemble d'objets sémantiques d'utilité publique (ex : entier, réel, couleur, temps, date, etc.) ;
- Un ensemble de présentations pour ces objets ;
- Un ensemble de types de tâches de « choix », chacune représentant une catégorie (Choix d'un élément parmi un petit nombre, choix de plusieurs éléments, choix parmi un sous ensemble de N, etc.) ;
- Un ensemble de présentations du choix (à base de boutons radio, menus, listes, boutons cycliques, etc.). Ces présentations sont indépendantes de la façon de représenter les éléments à choisir.

Le rendu d'un choix dépend de la combinaison d'un rendu de la façon de choisir (Data Selector Presenters) et d'un rendu des éléments à choisir (Data Presenters). Si il y a M rendus possibles pour présenter un choix et N rendus possibles pour les éléments à choisir, alors on a M x N rendus possibles de l'ensemble. ACE fournit ainsi une variété de présentations bien plus riche que dans les boîtes à outils classiques. Il est remarquable que peu de Bà0 l'aient depuis égalé. La Figure II-13 illustre les possibilités de polymorphisme sur des exemples simples.

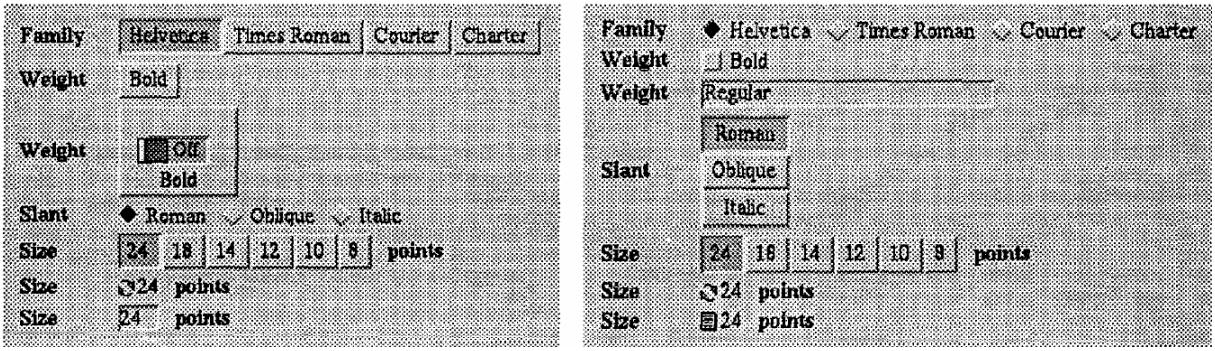


Figure II-13 : Polymorphisme du choix dans ACE.

Conclusion

ACE vise à dépasser les widgets classiques définis à trop bas niveau d'abstraction. Il propose de définir les interacteurs à un niveau proche du niveau concepts et tâches. Cependant, les opérateurs entre tâches et décorations de tâches ne sont pas pris en compte. Du point de vue de l'empreinte technologique, ACE s'appuie sur InterViews et ne semble pas couvrir d'autres technologies (HTML par exemple). Le Tableau II-3 caractérise ACE selon la grille d'analyse.

Tableau II-3 : Grille d'analyse pour ACE.

Niveau sémantique	Les interacteurs sont des concepts (couleur, etc.) ou des tâches (choix). L'assemblage des interacteurs est du niveau d'un modèle de dialogue puisque les opérateurs entre tâches n'existent pas en tant que tel. Le dialogue est codé explicitement.
Empreinte technologique	Le langage est C++ ; la BàO InterViews.
Extensibilité	Il est possible d'ajouter dynamiquement des Presenters aux concepts et tâches.
Malléabilité	Les Presenters sont interchangeables dynamiquement.
Contrôlabilité / Prévisibilité du rendu	Le rendu est totalement prévisible et contrôlable.

II.1.B. WAHID

WAHID [57 Jabarin 2003] se place explicitement dans le domaine de la plasticité au niveau widget. WAHID propose des widgets polymorphes s'adaptant à la plate-forme d'exécution. Les auteurs pensent que ce type d'approches est moins puissant qu'une plasticité « à la main » qui permet au concepteur un contrôle total de l'adaptation. Toutefois l'approche widget plastique est intéressante, du point de vue du programmeur, puisqu'elle préserve son savoir-faire. Il suffit d'utiliser la BàO WAHID dont l'API est calquée sur la Microsoft Foundation Classes (MFC) [87 Prosise 1999].

L'appui de WAHID dans la MFC lui confère une originalité : la plasticité, à l'exécution, d'IHM à l'origine non plastiques, mais construites à l'aide de la MFC.

WAHID propose deux architectures logicielles pour prendre en charge la plasticité des widgets : l'une interne, basée sur une BàO de widgets plastiques ; l'autre externe, qui permet de coupler des widgets plastiques à des applications déjà existantes dont les sources ne sont pas disponibles.

Architecture interne

Le but principal de « l'architecture interne » est de permettre le développement d'applications à base de widgets plastiques, capables de s'adapter à la plate-forme d'exécution. WAHID pose deux requis supplémentaires :

- Le développement d'une application plastique ne doit pas être plus compliqué que celui d'une application classique.
- Le développement d'une application plastique doit se faire à l'aide des outils et méthodes auxquels les développeurs sont accoutumés.

WAHID atteint le premier but en offrant une boîte à outils dont les widgets s'adaptent automatiquement à la plate-forme lors du lancement de l'IHM. Le second but est atteint en se basant sur MFC, une bibliothèque d'interacteurs architecturés selon MVC et largement utilisée dans le monde windows. WAHID spécialise les classes MFC et maintient la compatibilité avec le Wizard de VisualC++ utilisé par les concepteurs pour spécifier des IHM.

La méthode consiste à nommer les classes WAHID en suffixant celles de MFC par la lettre 'P'. Une simple substitution des noms de classes dans le code rend alors plastique l'application. Le surcoût de développement est donc très faible.

La scroll bar Horseshoe illustre l'approche (Figure II-14). Elle est conforme à l'API de la scroll bar MFC. Le programmeur n'a aucun surcoût à l'utiliser.

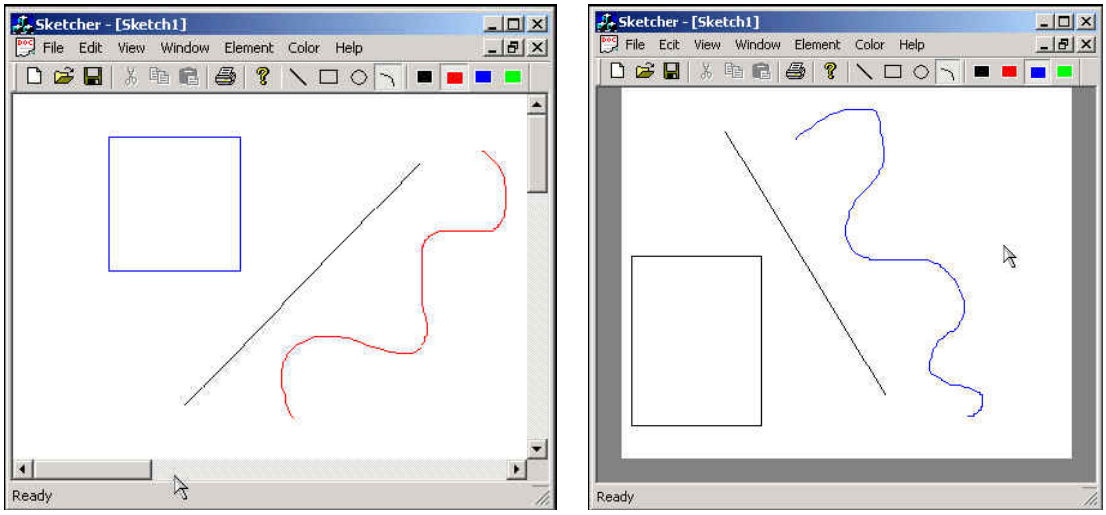


Figure II-14 : A gauche, la scroll bar classique. A droite, la scroll bar Horseshoe.

Selon [77 Myers 2000], WAHID offre un *low threshold*. Le *ceiling*, par contre, n'est pas élevé : la plasticité se limite à une substitution de présentation sur des plates-formes classiques (PC de bureau, tableau tactile ou tablette PC). Cette approche ne permet pas des réarrangements d'éléments à l'écran, encore moins de revoir toute la conception de l'IHM. Selon les auteurs, pour pouvoir prendre en charge une adaptation à des plates-

formes de type PDA ou smart phone, caractérisées par un faible nombre de pixels, il faudrait pouvoir spécifier des adaptations au niveau applicatif et non plus widget.

La section suivante décrit l'architecture externe pour une plasticité a posteriori d'IHM existantes.

Architecture externe

L' « architecture externe » a comme objectif de rendre une IHM existante plastique sans retoucher à son code. Ceci est précieux si justement le code n'est pas disponible. La plasticité se fait par ajout de nouveaux widgets, sans qu'ils se substituent aux anciens. Ces widgets ne peuvent pas s'insérer dans l'IHM, au mieux peuvent-ils être superposés à l'IHM, tel le menu en fleur par exemple (Figure II-15). Le menu classique ne disparaît pas pour autant. L'architecture externe permet une représentation multiple a posteriori, sans modifier le code applicatif.

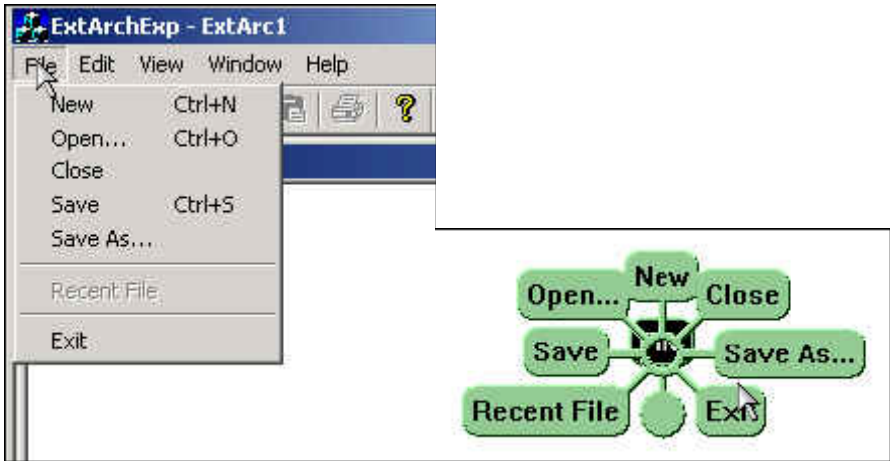


Figure II-15 : L'architecture externe permet une représentation a posteriori multiple.

Conclusion

En synthèse, l'architecture interne proposée dans WAHID est spécifique à la MFC. Cependant, le principe peut être étendu à n'importe quelle BàO. Le concepteur de telles BàO devra faire face à des problèmes techniques comme substituer un widget par un autre de façon transparente pour l'application. Du point de vue du programmeur (l'utilisateur de ces BàO), le surcoût lié à la plasticité sera pratiquement inexistant.

L'architecture externe permet d'ajouter des widgets simulant ceux de l'IHM d'origine (ex : menu en fleur, HorseShoe affiché dans une fenêtre à part…) sans que l'IHM soit, à l'origine, prévue pour (elle doit cependant respecter les standards de la MFC). Les widgets ne sont pas insérés dans l'IHM. Ils sont ajoutés « au dessus » de celle-ci. Ici encore, la transposition à d'autres BàO parait faisable.

Le pari de WAHID semble réussi : les requis posés par les auteurs sont respectés. Le programmeur peut continuer à utiliser sa BàO favorite et les outils associés. Il n'y a pratiquement pas de surcoût par rapport à un développement classique.

Le principal problème de WAHID réside, en réalité, dans la « faiblesse » des BàO visées (la MFC). Ces BàO n'offrent pas un niveau sémantique suffisant. Il est, par exemple, difficile de transformer un menu déroulant en boutons radio ou en liste déroulante. En effet, rien n'exprime que ces widgets ont tous pour but de faire un choix parmi un ensemble d'éléments. Leurs API peuvent, par ailleurs, être sensiblement différentes. Ce problème peut se résumer au fait que les widgets ne représentent pas les tâches utilisateur, mais des façons de réaliser une tâche, ces façons pouvant parfois être

incohérentes entre elles. Les auteurs reconnaissent la faiblesse d'une adaptation au niveau widgets :

- Il n'y a aucun contrôle possible de la part de l'utilisateur ou du programmeur.
- On ne peut pas agir sur une composition de widgets mais seulement sur un widget.
- On ne peut pas changer la mise en page. Il n'existe pas de widgets incarnant les opérateurs entre tâches. Par exemple, l'entrelacement pourrait être rendu au travers de menus, d'onglets, d'espaces les uns sous les autres etc. Si de tels widgets existaient, il serait alors possible d'agir sur la mise en page.

Le Tableau II-4 caractérise WAHID selon la grille d'analyse.

Tableau II-4 : Grille d'analyse pour WAHID.

Niveau sémantique	Les interacteurs sont du même niveau que la MFC, c'est-à-dire au niveau CUI/FUI.
Empreinte technologique	La BàO est la MFC (Microsoft Windows).
Extensibilité	Architecture interne : Les capacités de l'interacteur peuvent être étendues à la conception mais pas à l'exécution. Architecture externe : De nouvelles présentations peuvent être ajoutées dynamiquement à un interacteur déjà existant.
Malléabilité	Le choix de la présentation se fait automatiquement au démarrage.
Contrôlabilité / Prévisibilité du rendu	Ni le concepteur ni l'utilisateur ne peuvent changer la présentation. Si la plateforme est connue, le rendu est prévisible.

II.1.C. XFORMS

XFORMS est un standard du W3C visant à supplanter HTML pour la conception de formulaires sur le web. Le constat est que les formulaires HTML mélangent les données et la forme. XFORMS spécifie que les données et la présentation doivent être séparées dans des sections différentes. La partie « données » est fortement typée grâce à l'utilisation des XML-Schema. La norme XForms a été dès le début conçue pour enrichir la gestion d'un formulaire tout en le rendant indépendant du support. Les principaux atouts de XFORMS sont sa réutilisabilité (on peut créer plusieurs formulaires par page Web, ou bien un seul pour plusieurs pages), son faible besoin de programmation (la validation de certaines données est faite par le formulaire lui-même) et son indépendance vis-à-vis de la plate-forme. La spécification prévoit le support sur ordinateur, terminaux mobiles portatifs (Palm, PocketPC, téléphone...), télévision, imprimante et scanner. Sur imprimante et scanner par exemple, un utilisateur devra pouvoir imprimer un formulaire, le remplir à la main, et scanner le résultat pour le renvoyer sur le réseau au même titre qu'un formulaire WEB.

La Figure II-16 illustre la philosophie de XFORMS. Avec XFORMS, le concepteur décrit un formulaire contenant des types de données à saisir (texte, numéro, choix, etc.) : c'est la partie fonctionnelle de l'application (le XForms Model). Un autre langage (XForms User Interface) est utilisé pour décrire comment, selon le contexte (plate-forme et préférences du navigateur), le formulaire sera rendu à l'utilisateur. Le

formulaire ne sera pas présenté de la même façon selon qu'il sera rendu sur un ordinateur de bureau ou sur un téléphone. Les XForm Models peuvent être rendus dans d'autres langages que le XForms User Interface : par exemple, directement en XHTML ou WML.

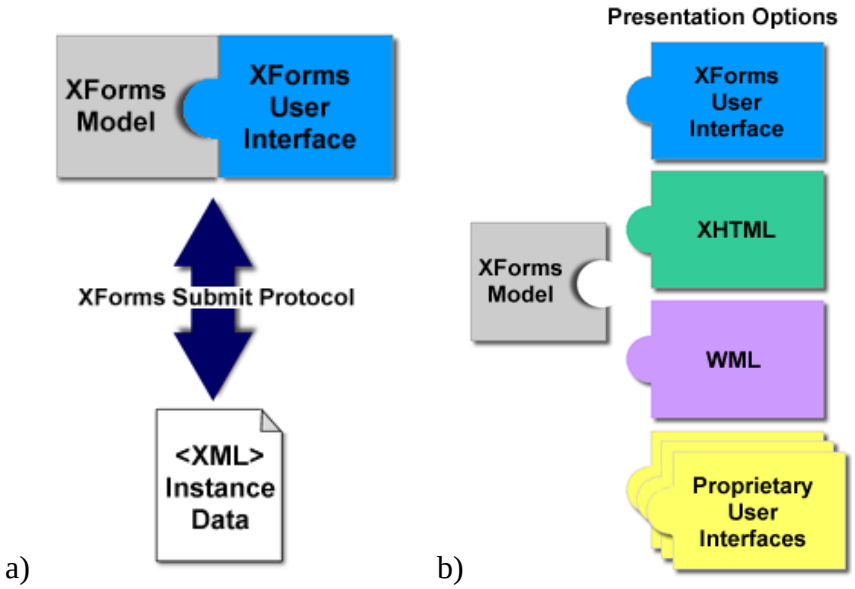


Figure II-16 : a)XForms Model définit la structuration des donnés XML manipulées par le formulaire. XForms User Interface définit la façon de les présenter. b) d'autres langages peuvent être utilisés pour assurer la présentation des formulaires XFORMS (ex : XHTML, WML, etc.).

Je m'intéresse ici plus particulièrement au langage de présentation XForms User Interface (XFormsUI). Il s'agit d'un langage XML visant à remplacer les balises liées au formulaire du XHTML. Ces balises sont de trop bas niveau sémantique. Les balises XFormsUI sont orientées tâches, contrairement aux balises HTML, plutôt orientées présentation. La Figure II-17 détaille les balises XFormsUI et leurs correspondants HTML.

XFormsUI	HTML	Description
input	<input type=« text »>	Champ de texte court
textarea	<textarea>	Champ de texte long
secret	<input type=« password »>	Texte confidentiel
output	SANS EQUIVALENT	Affichage de valeur
range	SANS EQUIVALENT	Sélection d'intervalle
select1	<select> ou <input type="radio">	Sélection unique
select	<select multiple="multiple"> ou <input type="checkbox">	Sélection multiple
upload	<input type=« file »>	Chargement de fichier ou de périphérique
trigger	<button>	Déclencheur d'événement
submit	<input type=« submit »>	Envoi du formulaire

Figure II-17 : Correspondances entre les balises XFormsUI et les balises HTML.

Les balises XFormsUI ne sont pas associées à une présentation particulière. C'est à l'interpréteur XForms (en pratique le navigateur Web) qu'incombe de trouver une

présentation adaptée à la tâche (ex : select1) ET au concept (ex : date). Un choix simple d'éléments non typés peut classiquement être représenté par le navigateur sous la forme de boutons radio ou menu déroulant (Figure II-18).

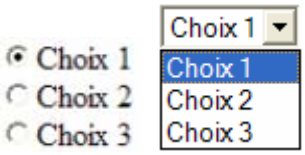


Figure II-18 : Deux représentations possibles d'un choix simple pour un navigateur.

Un choix multiple d'éléments non typés peut classiquement être représenté par un navigateur sous la forme de cases à cocher ou d'une liste d'éléments dont les sélections sont surlignées ou cochées (Figure II-19). La propriété *appearance* permet de spécifier à l'interpréteur si tous les choix doivent être proposés : tous (full), un certain nombre avec possibilité de navigation (compact) ou un minimum de choix avec la possibilité d'afficher de façon temporaire les autres (minimal) (Figure II-19).

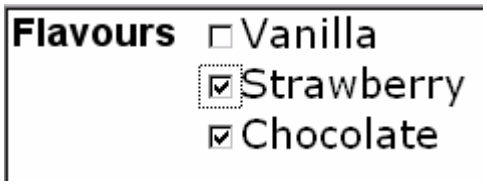
appearance="full"	appearance="compact"	appearance="minimal"
		

Figure II-19 : Différentes présentations du choix selon la balise appearance fixée par le concepteur.

La Figure II-20 illustre des présentations possibles pour un spécificateur de date. L'utilisateur peut soit entrer la date au clavier, soit la choisir dans un calendrier. Le choix de la présentation est fait par le navigateur.

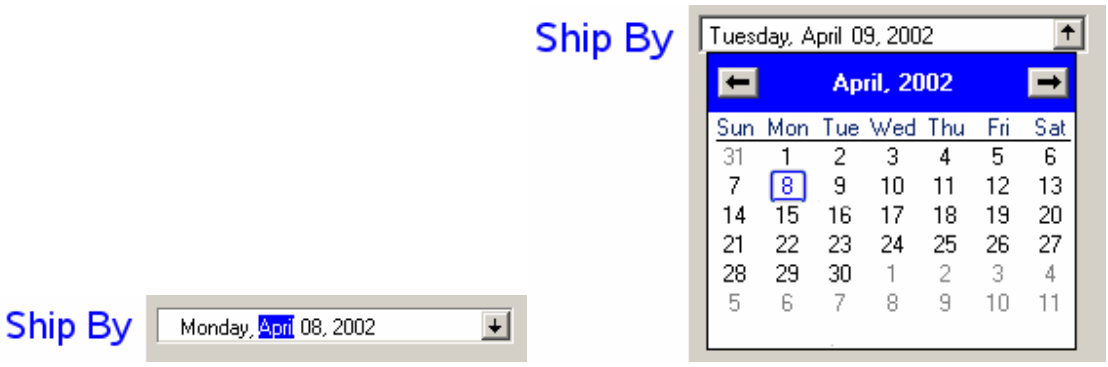


Figure II-20 : Différentes présentations pour un spécificateur de date.

Conclusion

XForms a été conçu en ayant à l'esprit que le WEB est et sera de plus en plus accessible via des terminaux aux caractéristiques variées (PDA, téléphones, télévisions, PC de bureaux, imprimantes, fours, etc.). Aussi, les parties « structure des données » et « présentation » doivent-elles être séparées et les balises orientées tâches et non plus présentation. Le polymorphisme est à la charge des interpréteurs (les navigateurs) qui devront sélectionner une présentation adéquate en fonction de la plate-forme cible, de la tâche et du concept manipulé.

XForms ne couvre que les IHM de type formulaire. La critique se limite donc à ce cadre :

- XForms ne modélise pas les opérateurs entre tâches. Il ne semble pas possible de modéliser directement en XForms une séquence ou un entrelacement (qui pourrait être rendu au travers d'onglets par exemple).
- XForms ne laisse aucune possibilité au concepteur de préciser quelles sont les présentations souhaitées. L'intégration de widgets sur mesure n'est pas non plus possible.
- Les présentations dépendent du navigateur utilisé. Il est à craindre qu'elles soient substantiellement différentes d'un navigateur à l'autre. Par exemple, tous les navigateurs ne représenteront peut-être pas, par un calendrier, le choix d'un jour de l'année, y compris sur des plates-formes similaires (par ex : FireFox et Internet Explorer sur PC).

Tableau II-5 : Grille d'analyse pour XForms.

Niveau sémantique	Les balises XFormsUI représentent des tâches utilisateur.
Empreinte technologique	Le WEB.
Extensibilité	Le langage est standardisé et non extensible par un concepteur.
Malléabilité	Les styles CSS sont applicables.
Contrôlabilité / Prévisibilité du rendu	Le rendu de l'interacteur est à la charge de l'interpréteur XForms. Il n'est pas contrôlable par le concepteur. Celui-ci peut seulement parfois donner des indications de rendu (ex : l'attribut « appearance » de la balise select). Il est potentiellement imprévisible puisqu'il pourra varier d'un interpréteur à l'autre.

II.2. Dominante Multimodalité

Dans cette catégorie, les rendus impliquent plusieurs modalités au sens humain (vision, audition, etc.). Les BàO étudiées sont FRUIT et Multimodal Widgets.

II.2.A. Fruit

Les auteurs de FRUIT [61 Kawai 1996] proposent d'abstraire les widgets classiques. Ils constatent que ces widgets gèrent à la fois les aspects sémantiques et présentationnels. Ils proposent de séparer ces préoccupations avec, d'un coté, les widgets abstraits qui gèreront la sémantique et, de l'autre, les widgets concrets qui gèreront le rendu.

Les auteurs identifient trois grandes classes de widgets abstraits (AW pour Abstract Widget) :

- Les AW de base modélisent une interaction basique comme l'activation, la saisie de texte. Ces AW sont en lien direct avec l'utilisateur.
- Les AW container sont chargés d'organiser d'autres AW, par exemple de gérer la mise en page des IHM graphiques (GUI pour Graphical User Interfaces).
- Les AW composés groupent des AW dont chacun peut influer sur l'autre. C'est le cas, par exemple, d'un sélecteur de fichiers. Les AW composés sont notamment utiles pour atteindre le niveau sémantique désiré.

Les AW de base sont eux-mêmes organisés en quatre grands groupes :

- Command : Ils se matérialisent généralement dans les GUI par des boutons. Ils peuvent être aussi des commandes saisies au clavier ou prononcées.
- Selection : En GUI, ce sont les listes, les boutons radio, les menus, etc. En vocal, ils peuvent se matérialiser par des mots ou des numéros.
- Valuator : En GUI, ils se matérialisent classiquement par des potentiomètres (type slidebar), des jauges, etc. Dans le cas d'IHM textuelles, l'utilisateur peut saisir une valeur au clavier. Il peut aussi l'énoncer oralement dans le cadre d'IHM vocales.
- TextDisplay, TextInput : En GUI, ce sont les libellés et champs texte classiques. En vocal, c'est toute prononciation système ou humaine.

Les AW container sont eux aussi classés en trois grands groupes :

- Shell : Dans les GUI, ce sont les widgets « toplevel », qui fournissent le contexte graphique aux autres, c'est-à-dire le plus souvent les fenêtres. Dans le cadre d'IHM textuelles, ce sont les prompts.
- Menu : C'est un type de shell qui prend temporairement le focus.
- Group : Les groupes gèrent la distribution du focus à leurs fils.

Enfin, les AW composés étant définis à la guise du concepteur, ils n'ont pas de classement particulier. A titre d'illustration, la Figure II-21, extraite de [61 Kawai 1996], met en correspondance les widgets TK et AW. On peut noter qu'un widget AW peut avoir plusieurs représentations TK.

Tk widget	Abstract widget
toplevel	shell
frame	group
label	textdisplay
message	textdisplay
button	command
checkbutton	command
radiobutton	selection
listbox	selection
scrollbar	valuator
scale	valuator
entry	textentry
Text	textentry
Menu	Menu
Canvas	group

Figure II-21 : Correspondances entre widgets TK et AW.

Techniquement, une application FRUIT se compose de trois parties (Figure II-22) :

- Les widgets rendus sont répartis dans un ou plusieurs shells d'interaction. Généralement, l'utilisateur choisit un seul shell (par exemple vocal), mais il peut décider d'en lancer plusieurs (par exemple graphiques) pour compléter.
- Les widgets abstraits centralisent la logique de l'application. Ils interprètent au niveau de l'application les opérations faites par l'utilisateur sur les widgets rendus.
- Le « sessions manager » est l'infrastructure sous jacente à FRUIT.

Le programmeur construit l'IHM à l'aide de widgets abstraits. L'utilisateur agit au travers de widgets rendus dans un ou plusieurs shells d'interaction (graphique, vocal, etc.). Ces widgets rendus communiquent aux widgets abstraits, via un protocole d'interaction, les opérations effectuées. Un « session manager » gère les applications FRUIT (Figure II-22).

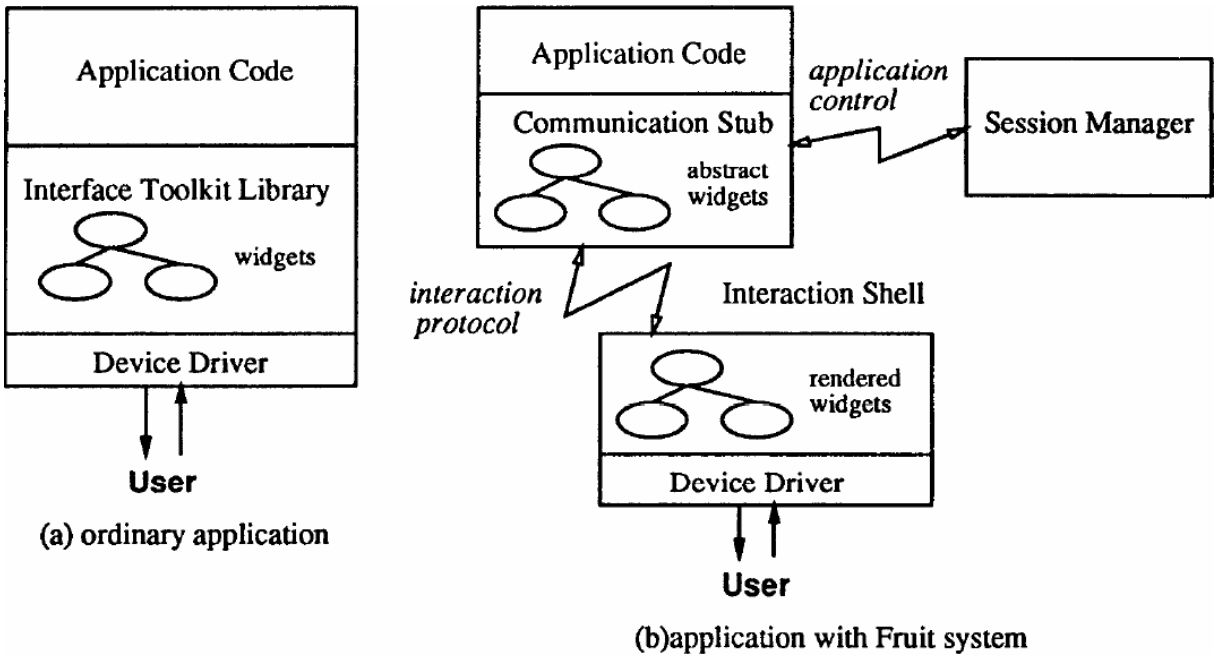


Figure II-22 : Comparaison entre une application classique (a) et une application FRUIT (b).

Le choix de la présentation d'un widget est réalisé par une boîte noire qui raisonne notamment en fonction de l'AW correspondant et de l'interaction shell. Ce choix de l'interacteur est du ressort du système seulement. La mise en page (dans le cas graphique) est aussi laissée à la charge de cette boîte noire.

Conclusion

FRUIT découple la sémantique de l'application de sa présentation. FRUIT s'appuie sur les BàO existantes (ex : TK) pour le rendu des widgets abstraits. Les mécanismes d'interaction n'ont pas à être reprogrammés. La notion de shell permet de définir dans quelle technologie seront rendus les widgets et incidemment dans quelle(s) modalité(s). Les auteurs insistent sur le fait que des IHM vont ainsi pouvoir être rendues simultanément en graphique, en vocal et en mode texte. La notion de shell permet d'envisager le rendu simultané des widgets dans différentes BàO, ce qui pourrait permettre d'avoir des rendus graphiques traditionnels WIMP, Web, post-WIMP, 3D, etc. De même, en vocal, on peut imaginer différents shells de reconnaissance vocale plus ou moins performante ou adaptée au contexte d'usage (langue de l'utilisateur par exemple). L'approche a toutefois des limites :

- Bien que la notion de groupe apparaisse, ni les opérateurs entre tâches, ni les décorations ne sont modélisés dans FRUIT. Il est donc impossible de dire que des AW doivent être entrelacés, en séquence, en disjonction, etc. Cela limite les possibilités de rendu et oblige, en réalité, le concepteur à câbler ces opérateurs dans le graphe d'AW.
- FRUIT ne propose pas de système pour sélectionner les types de rendu associés aux AW. Il semble que cela soit laissé à la discrétion de l'interaction shell.

Tableau II-6 : Grille d'analyse pour FRUIT.

Niveau sémantique	Les Abstract Widgets sont de niveau tâche. Ils sont indépendants de la modalité.

Empreinte technologique	En termes de modalités, FRUIT couvre les IHM textuelles, graphiques avec TK, vocales. Un même AW peut être rendu simultanément selon plusieurs de ces technologies.
Extensibilité	L'extensibilité n'est pas possible au niveau de l'interacteur. L'extensibilité n'est possible qu'au niveau du shell. C'est-à-dire que pour enrichir les présentations d'un abstract widget, il faut enrichir le shell.
Malléabilité	Il n'y a pas de transformation applicable au niveau de l'AW.
Contrôlabilité / Prévisibilité du rendu	Le rendu de l'AW n'est contrôlable qu'en choisissant un shell pour le rendu. Le rendu à l'intérieur d'un shell est prévisible.

II.2.B. Multimodal Widget

Constatant que les situations d'interaction peuvent être variées (à l'intérieur, dehors, environnement calme/bruyant, seul/en groupe, etc.), l'auteur identifie la nécessité pour une application de pouvoir être rendue perceptible à l'utilisateur au travers de différents sens humains (visuel, auditif, tactile, etc.). L'auteur propose une BàO d'interacteurs multimodaux (mettant en jeu différents sens humains) visant à faciliter la construction de telles applications. Les multimodal widgets [30-31Crease 2000-2001] répondent à quatre requis :

- Un widget doit pouvoir être rendu simultanément dans plusieurs modalités.
- Un widget doit pouvoir sélectionner la modalité la plus adaptée au contexte d'usage et limiter l'utilisation d'une modalité n'ayant pas les ressources nécessaires.
- Le feedback d'un widget doit pouvoir être facilement changé pour une ou plusieurs modalités sans que cela ait des effets sur les autres.
- Les feedbacks des différentes modalités doivent être cohérents non seulement pour un même widget mais aussi entre widgets.

Du point de vue du développeeur, la BàO des multimodal widgets reprend l'API de la BàO SWING. Les habitudes du programmeur sont ainsi préservées. Par exemple, pour créer un bouton sous SWING et l'ajouter à un panel, le code est le suivant :

```
JButton button = new JButton("Progress"); panel.add(button);
```

Avec les multimodal widgets, l'exemple équivalent est le suivant :

```
MButton button = new MButton("Progress"); panel.add (button.getTheWidget());
```

La Figure II-23 fournit l'architecture d'une application utilisant les multimodal widgets. Le *feedback controller* assure que les modalités sont traitées d'égales à égales et que chaque widget est multimodal. Il traduit les événements reçus en termes indépendants de la modalité et les transmet aux *modality mappers.*

Le *ressource manager* assure la connaissance des ressources disponibles et de leur adéquation à l'environnement. Il reçoit des informations de la part de trois modules : le *control panel* (préférence de l'utilisateur quant à la modalité à utiliser) ; les *output modules* (indiquent si les ressources sont suffisantes pour rendre les widgets via une modalité, compte tenu de l'importance accordée par l'utilisateur à celle ci) et d'autres applications (via son API, ces applications peuvent renseigner le système sur l'état de l'environnement).

Les *outputs modules* et le *control panel* assurent en outre que le feedback des widgets pourra être facilement changé. En effet, les outputs modules peuvent facilement être remplacés au travers du control panel.

Enfin, la cohérence de l'ensemble est assurée par le *rendering manager*. Le rendering manager peut modifier un son pour qu'il n'interfère pas avec un autre (en changeant le timbre par exemple). De même, il assurera qu'un bouton texturé comme du bois n'est pas associé à un son métallique.

La Figure II-24, extraite de [30 Crease 2000] illustre le comportement de l'architecture dans le cas d'une souris survolant la représentation graphique d'un bouton. Il s'agit d'associer un feedback à cet événement : il combine ici le graphique et le vocal.

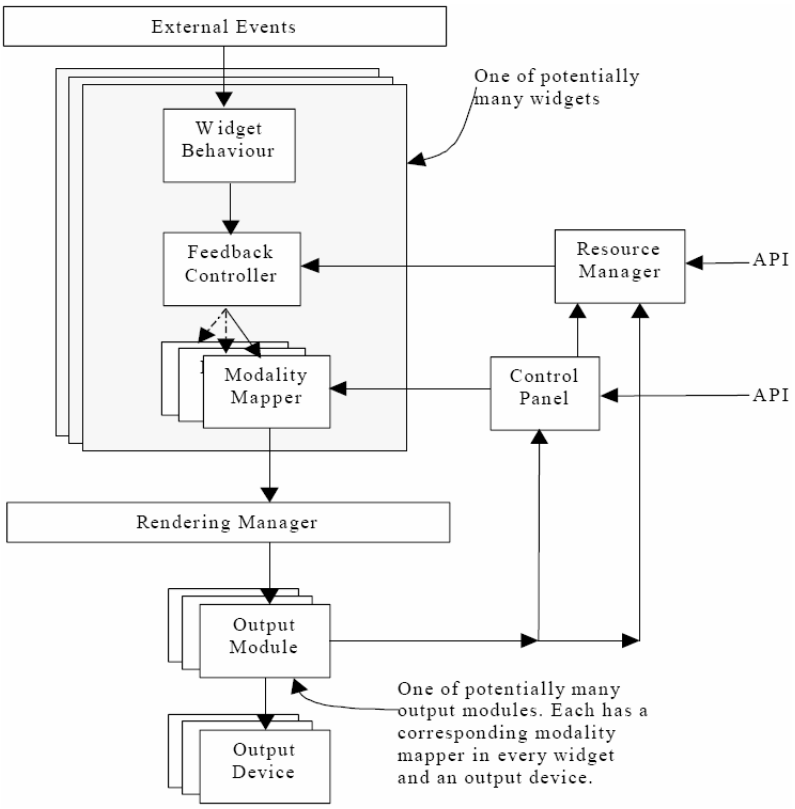


Figure II-23 : Architecture logicielle de la BàO « Multimodal widgets ».

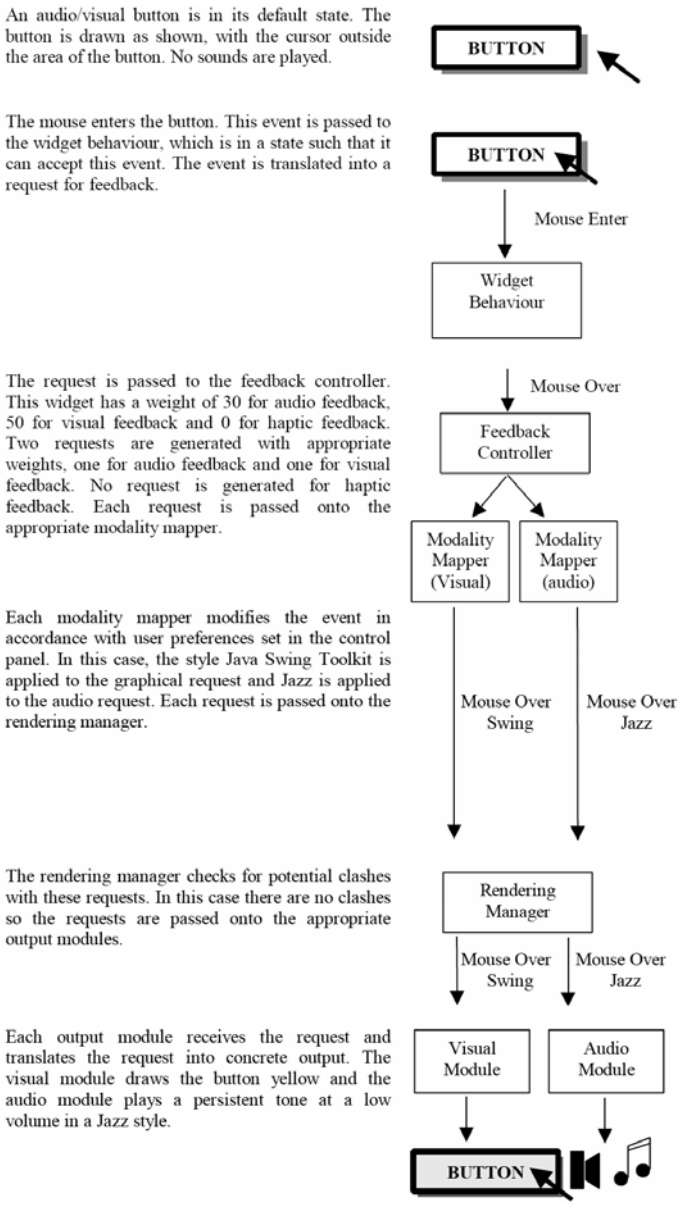


Figure II-24 : L'architecture en action sur l'exemple du feedback lors du passage de la souris sur un bouton.

La Figure II-25 extraite de [30 Crease 2000] montre différents rendus graphiques pour une jauge. La sélection du rendu se fait en fonction, d'une part, de l'importance donnée à la modalité graphique pour ce widget et, d'autre part, des ressources écran disponibles. C'est au concepteur de widgets de fournir les différentes présentations.

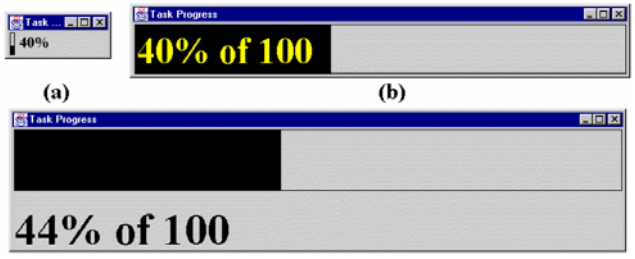


Figure II-25 : Différentes présentations d'un même widget "jauge". Elles consomment plus ou moins de surface d'affichage.

Conclusion

Les multimodal widgets offrent une BàO permettant au concepteur de construire simplement des IHM qui pourront être rendues dans plusieurs modalités (c'est-à-dire mettant en jeu différents sens humains). Le style de programmation est très proche de SWING. La surcharge par rapport à SWING est pratiquement nulle puisque le programmeur n'a pas à se soucier du rendu multimodal. Il est pris en charge par le système. La cohérence entre widgets et entre modalités est assurée par le rendering manager. Le choix de privilégier telle ou telle modalité est, en partie, laissé à l'utilisateur via le control panel.

Toutefois, en voulant rendre la programmation d'IHM multimodales aussi proche que possible de la programmation SWING, les multimodal widgets se trouvent confrontés aux limites de SWING en terme d'expressivité des IHM :

- Les opérateurs entre tâches ne sont pas modélisés. Les tâches ne le sont pas vraiment.
- Les widgets n'ont pas de sémantique explicite. On peut ainsi douter que le rendering manager sera capable de maintenir des cohérences du type « tous les widgets associés au concept C1 doivent être cohérents entre eux et différenciables de ceux associés au concept C2 ».
- L'exemple de la jauge semble montrer qu'un certain polymorphisme est possible au sein d'une même modalité. Cependant, c'est au concepteur de l'output module de ce widget de fournir ces présentations. Il semble alors difficile pour un autre concepteur d'ajouter de nouvelles présentations à un output module déjà existant.
- Les outputs modules graphiques sont intrinsèquement liés à SWING. Il semble difficile de changer de BàO, par exemple pour créer du post WIMP.

Tableau II-7 : Grille d'analyse des Multimodal Widgets.

Niveau sémantique	Les multimodal widgets sont plus une architecture qu'une BàO. Il n'y a que deux widgets : le bouton et la jauge. Il est donc difficile de caractériser le niveau sémantique. Toutefois, on peut dire que les widgets sont au moins de niveau AUI puisqu'ils n'ont pas de représentation intrinsèque.
Empreinte technologique	Les widgets peuvent être rendus suivant différentes modalités dans différentes technologies. Cela est illustré avec un rendu vocal et un rendu graphique.
Extensibilité	Il est possible d'étendre les rendus via l'ajout d'output modules et de modality mappers.
Malléabilité	Il est possible de changer dynamiquement d'output modules.
Contrôlabilité / Prévisibilité du rendu	Le contrôle est semi automatique. L'utilisateur pondère les modalités à l'aide du « control panel ». Le système modifie le rendu en conséquence. Le rendu est dépendant des output modules, ceux-ci pouvant implémenter plusieurs rendus.

II.3. Dominante Post-WIMP

Les BàO ici étudiées n'ont pas été développées pour la plasticité. Je les examine toutefois au cas où certaines de leurs originalités pourraient faciliter la conception d'IHM plastiques. Sous ce terme de Post-WIMP (Windows Icon Mouse Pointer), se cachent des travaux visant à dépasser les limites des BàO « classiques » (de type

SWING) dont les fondements théoriques datent des années 80 et se justifiaient par les contraintes de l'époque (ordinateur de bureau, écran, pointeur, clavier, interfaces formulaires). Les BàO examinées sont Ubit et Attach Me Detach Me.

II.3.A. UBIT

Considérant les BàO « classiques », E. Lecolinet constate que [65-66 Lecolinet 1999 et 2003] :

- Les BàO sont artificiellement figées. Seules les IHM de type WIMP sont faciles à réaliser. Elles posent des contraintes qui n'ont plus lieu d'être.
- Il existe une séparation artificielle entre les IHM WEB et non WEB. Il n'y a pas de raison que la programmation d'une IHM pour le WEB soit si différente. Cette séparation s'explique par des raisons historiques. Elle doit être surmontée.
- De nombreuses personnes se disent, et sont, capables de coder de petites IHM en HTML. Inversement, très peu s'estiment, et sont capables, de coder des IHM à l'aide des BàO classiques (SWING, QT…).

Le but d'UBIT est de dépasser ces divisions et de permettre au programmeur de facilement définir de nouveaux widgets. Je ne m'intéresse pas ici aux différences entre UBIT et les BàO classiques, mais aux caractéristiques d'UBIT qui pourraient faciliter le développement d'IHM plastiques. Je repére le contexte généralisé, l'héritage, la notion de style et la structure en Directed Acyclic Graph (DAG).

Propriétés utiles pour la plasticité

Une IHM UBIT (comme d'ailleurs une IHM SWING ou HTML) peut être vue comme un graphe orienté où chaque nœud représente un élément d'interface et chaque relation entre deux nœuds représente la notion « contient ». L'affichage de l'interface se fait relativement au parcours de ce graphe. Chaque nœud est affiché en fonction de son père et en fonction d'un contexte graphique courant. Par exemple, si le contexte graphique courant indique que la couleur est noire, alors un texte sera affiché en noir. Cette notion de contexte graphique vient du monde de la synthèse d'image et des graphes de scène. Ce contexte est généralement réduit à une transformation spatiale, la couleur, la texture, etc. UBIT propose de généraliser cette notion en y incorporant potentiellement n'importe quelle information, par exemple la fonte, mais aussi des informations spécifiques à une application. La Figure II-26 illustre le rendu d'un graphe UBIT représentant un bouton.

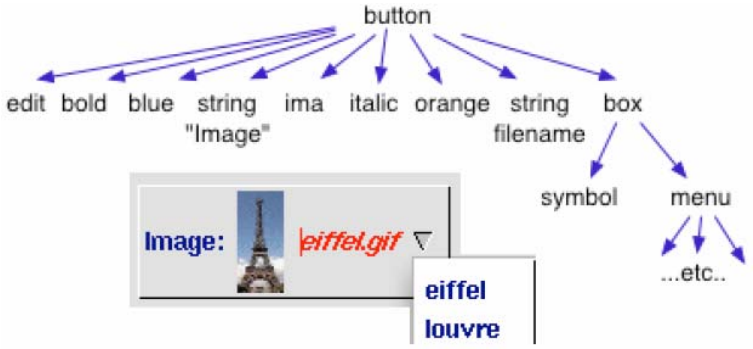


Figure II-26 : Graphe UBIT représentant un bouton. Des éléments du graphe comme "bold" ou "blue" modifient le contexte ainsi le texte « Image » est-il affiché en bleu gras.

Les informations de style (fonte, couleur, etc.) placées dans le contexte sont transmises lors du parcours du graphe UBIT. Ainsi, pour modifier la fonte utilisée par défaut dans toute l'application, il suffit d'ajouter une information de style dans le contexte généralisé du noeud racine de l'application.

Un style global peut être donné en spécifiant des valeurs pour la fonte, la couleur du texte, la couleur de fond, etc. Ces valeurs sont transmises aux descendants. Le style peut être surchargé à n'importe quel niveau du graphe. Cette façon de faire est à rapprocher des langages de style existants, par exemple, CSS pour le Web. Le style donné peut n'être limité qu'à des informations de mise en forme de type couleurs ou fonte, mais il est aussi possible de l'utiliser pour réaliser des modifications plus complexes, comme des zooms sémantiques (l'interface change en fonction du zoom).

Cette notion de style, de contexte généralisé, permet de modifier l'apparence d'une IHM. Couplé à une structure en DAG, elle permet de réaliser des vues multiples synchronisées.

Alors que les BàO classiques imposent une structure d'arbre au graphe de widgets, UBIT, s'inspirant des BàO 3D (graphe de scènes) et de certaines BàO 2D ([47 Fresco] [73 Linton 1994], [80 OpenInventor], [58 Jazz] [12 Bederson 2000], etc.) propose d'utiliser une structure en DAG. Cela permet à un élément d'avoir plusieurs pères et donc d'être partagé (Figure II-27).

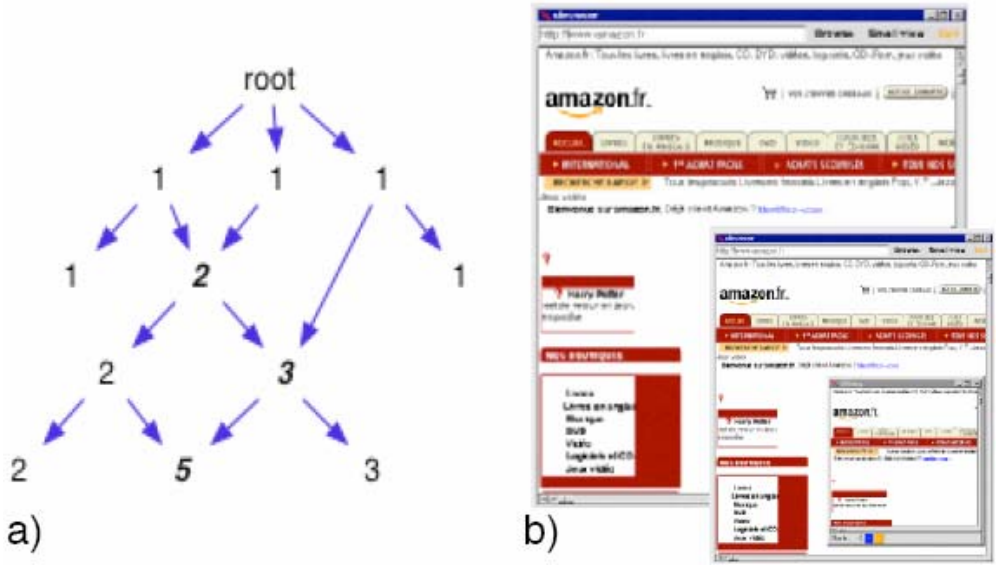


Figure II-27 : a) Structure en DAG (Directed Acyclique Graph). Les numéros représentent le nombre de fois que le nœud sera affiché. b) Une même interface affichée deux fois, avec différents facteurs de zoom. Le facteur de zoom est un élément du contexte.

La partage permet de réaliser facilement des vues multiples : il suffit d'hériter de plusieurs pères. Les différences entre les vues seront assurées par des contextes différents. Par exemple, le même bouton pourra être affiché deux fois, à des endroits différents et avec une couleur de fond différente (Figure II-28). Ce même principe peut être appliqué à la réalisation de vues radars, à la réplication d'interfaces, etc.

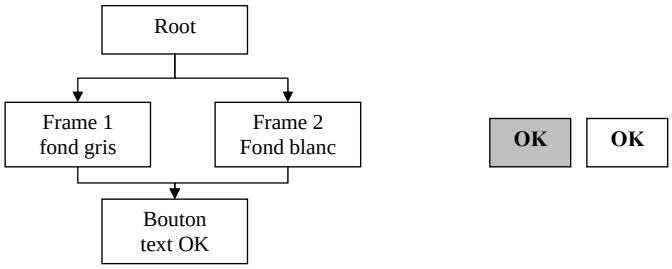


Figure II-28 : Exemple de vue multiple affichée selon différents styles propagés par le contexte généralisé. A gauche, le graphe montre un bouton ayant deux frames pour pères, chacune redéfinit la couleur courante. A droite, le résultat.

L'utilisation de cette structure de DAG permet de réaliser une certaine forme de polymorphisme La Figure II-29 montre l'utilisation de nœuds « Conditions » qui sont mutuellement exclusifs (un et un seul est vrai à la fois). Les conditions portent sur le contexte généralisé, par exemple la place disponible, les préférences utilisateur, des informations propres à l'application, etc. Si la condition est fausse, le sous DAG n'est pas affiché. Seule une forme est donc affichée à un instant donné. Les conditions I et N accèdent au même sous DAG : celui-ci sera affiché différemment en fonction des contextes I et N.

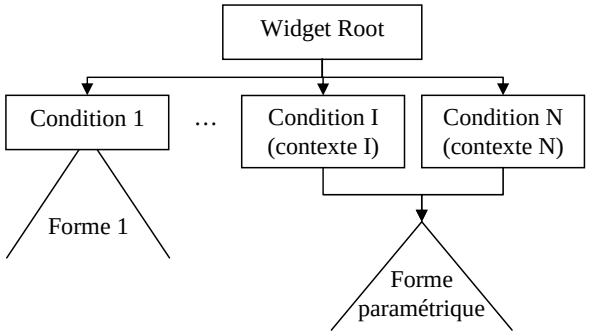


Figure II-29 : Exemple de réalisation de widget polymorphe.

Conclusion

UBIT permet de réaliser des IHM extrêmement flexibles, ne se limitant pas aux IHM standardisées (SWING, TK…). Cela facilite l'innovation. La notion de style, de contexte généralisé est un pas en direction d'une harmonisation entre les conceptions d'IHM Web et non Web. L'objectif est d'allier la facilité de développement en HTML et la puissance des BàO classiques. UBIT fournit des éléments extrêmement modulaires. La création de nouveaux widgets revient à construire des brickgets, assemblages de bricks. De nouvelles bricks peuvent être ajoutées par programmation, par exemple pour fournir un canevas 3D. Enfin, la structure en DAG permet de réaliser de façon aisée des vues multiples et offre une façon simple pour réaliser des widgets polymorphes.

Si UBIT est une BàO d'interacteurs innovante, elle reste au niveau CUI/FUI. Il n'y a pas de modélisation de l'interface au niveau tâche utilisateur par exemple.

Tableau II-8 : Grille d'analyse pour UBIT.

Niveau sémantique	CUI/FUI
Empreinte technologique	C++ / X11
Extensibilité	Extensibilité par ajout de bricks. Des interacteurs sur mesure sont faciles à définir.
Malléabilité	Elle est assurée par les bricks et le contexte généralisé.

Contrôlabilité / Prévisibilité du rendu	Le rendu de l'interacteur (brickget) est contrôlable via le contexte généralisé.

II.3.B. ATTACH ME DETACH ME

Ici, les interacteurs ne diffèrent pas des interacteurs des BàO WIMP. L'innovation vient de leur capacité à migrer d'une plate-forme à une autre. Les auteurs [52 Grolaux 2005] constatent que les IHM sont toujours centralisées alors que de plus en plus d'ordinateurs sont disponibles par utilisateur. Ils expliquent cela par le fait que les BàO ou les systèmes d'exploitation ne permettent pas de réaliser des programmes distribués de façon transparente pour le programmeur. Les auteurs introduisent le concept d'interface détachable, caractérisé par :

- La détachabilité : Tout élément d'interface doit pouvoir être détaché de son emplacement tout en continuant à fonctionner, sous condition que le concepteur l'ait autorisé.
- La migrabilité : Tout élément d'interface détachable doit être migrable sur une autre plate-forme, quelles que soient ses caractéristiques physiques et logicielles.
- La plastifiabilité : Tout élément d'interface migré doit s'adapter ou être adapté à la plate-forme cible. C'est la notion de remodelage.
- L'attachabilité : L'élément d'interface détaché doit pouvoir être rattaché à n'importe quelle interface.

La BàO proposée par les auteurs est construite sur l'environnement Mozart-Oz (www.mozart-oz.org). Cet environnement offre, de façon transparente, des capacités de système distribué. Les auteurs en profitent. Ils définissent des primitives pour offrir diverses formes de migration :

- Display(UI, ptf) : Les éléments de l'IHM UI sont affichés sur la plate-forme ptf.
- Undisplay(UI, ptf) : Les éléments de l'IHM UI sont effacés de la plate-forme ptf.
- Copy(UI, source, target) : Les éléments de l'IHM UI sont copiés de la plate-forme source vers la plate-forme target.
- Expose(UI, source, target) : Les éléments de l'IHM UI sont copiés de source vers target mais ne sont pas manipulables sur target.
- Return(UI, target, source) : Les éléments de l'IHM UI qui ont été migrés de source vers target sont renvoyés vers source.
- Transfer(UI, source, target) : Les éléments de l'IHM UI sont copiés de source vers target et supprimés de source.
- Delegate(UI, source, target) : Est équivalent à la succession de Transfert(UI, source, target) puis Return(UI, target, source).
- Switch(source UI, source, target UI, target) : Les éléments de source UI sur source et target UI sur target sont intervertis.

La Figure II-30 illustre les opérations de détachement et d'attachement sur l'IHM de l'application de dessin QTkDraw. La palette d'outils peut être détachée. Elle peut aussi être migrée sur une autre plate-forme.

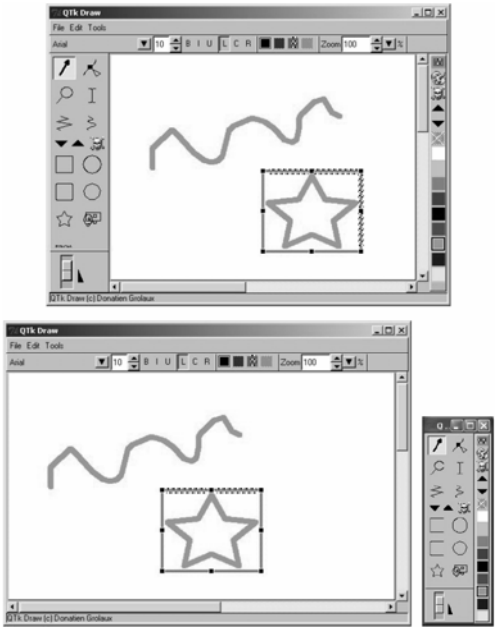


Figure II-30 : Les barres d'outils de l'éditeur de dessin peuvent être détachées et rattachées.

La BàO graphique est basée sur une surcouche de TK, QTK. Les IHM sont définies à ce niveau d'abstraction (CUI/FUI).

Conclusion

Les auteurs proposent une BàO assurant la migration d'IHM, en toute transparence pour le programmeur. Plusieurs primitives sont proposées pour contrôler la façon dont une IHM est migrée. En particulier, une IHM peut être affichée à distance sans qu'il ne soit possible de la manipuler (*Expose*). Ce travail montre qu'il est possible d'ajouter des capacités de migration à une application en s'appuyant sur une infrastructure adéquate.

Les auteurs mentionnent la plasticité dans leur concept d'interface détachable mais ne proposent pas de solution à ce problème (dans ce travail). Les auteurs travaillant par ailleurs sur UsiXML, il est probable qu'à terme les travaux soient intégrés et que des IHM UsiXML soient générées en utilisant cette BàO. Elles bénéficieraient alors des capacités de migration.

Tableau II-9 : Grille d'analyse pour Attach Me Detach Me.

Niveau sémantique	CUI
Empreinte technologique	Mozart-Oz et QTK
Extensibilité	Pas dynamique.
Malléabilité	Un interacteur peut être migré d'un processus à un autre.
Contrôlabilité / Prévisibilité du rendu	Le contrôle est total. Il en est de même pour la prévisibilité.

II.4. Conclusion sur les BàO

Suite à cet état de l'art, je constate qu'aucune des BàO étudiées ne répond pleinement aux requis fixés. Le Tableau II-10 rappelle nos requis en indiquant les points forts des différentes approches.

Tableau II-10 : Synthèse des BàO de l'état de l'art.

Niveau sémantique	Comme dans ACE, XFORMS ou FRUIT, les interacteurs doivent être de niveau tâche. Il faut ajouter à cela la possibilité de représenter n'importe quelle tâche et les opérateurs entre tâches. Aucune approche ne couvre les opérateurs entre tâches ni les décorations. Le graphe doit pouvoir être considéré selon différents points de vue : tâche, AUI, CUI, FUI.
Empreinte technologique	Comme dans FRUIT ou Multimodal Widgets, l'empreinte technologique doit être large. Le rendu doit pouvoir se faire selon différentes modalités et potentiellement dans différentes BàO (HTML, SWING, Ubit, Vocale, tactile…). Comme dans FRUIT, le graphe doit pouvoir être rendu simultanément dans plusieurs technologies.
Extensibilité	Comme dans les Multimodal Widgets ou ACE, le rendu d'interacteurs doit pouvoir être étendu dynamiquement, idéalement avec des rendus non prévus lors de la conception.
Malléabilité	La notion de style de XFORMS permet de paramétrer le rendu. Les notions de Presenters dans ACE ou d'output modules dans Multimodal Widgets permettent de réaliser un polymorphisme plus profond que les styles de XFORMS. La conception de l'IHM comme un graphe de scène, tel que présenté dans UBIT, permet d'envisager une grande souplesse pour transformer l'IHM de façon structurelle.
Contrôlabilité / Prévisibilité du rendu	Le rendu doit être contrôlable et prévisible. A cet égard, le contexte généralisé de UBIT et les CSS liés à XFORMS sont des pistes intéressantes pour contrôler le rendu.

La Tableau II-10 montre que les propriétés d'extensibilité, malléabilité et contrôle ne sont pas satisfaites du seul ressort des BàO. Les notions de styles, de contexte généralisé, de rendus non prévus à la conception sont certes liées aux BàO mais débordent de ce cadre. C'est pourquoi il m'a semblé intéressant d'avoir une compréhension de domaines connexes aux BàO, à savoir : les langages de transformation (pour la malléabilité et le contrôle) et les approches orientées services (pour l'extensibilité). C'est l'objet de la section suivante.

III. Transformations et services : des outils pour les BàO d'interacteurs plastiques

Transformations et services sont deux outils qui peuvent être pertinents pour la plasticité des IHM. En IHM, il existe plusieurs types de langages permettant de transformer une IHM. Je fais le point sur les langages de skin, les langages de style et les langages de transformation. J'examine, en particulier, la simplicité d'expression de ces langages et la puissance des transformations exprimables.

Je m'intéresse ensuite aux approches orientées service (SOA). Ces approches peuvent être pertinentes pour répondre au problème de l'imprévisibilité du contexte d'usage. Le domaine des SOA est vaste. Je me limite ici à l'étude des annuaires de service. Il s'agit d'examiner dans quelle mesure des services peuvent être capitalisables dans un annuaire, puis découverts. L'étude est générale, non limitée à des services pour l'IHM.

III.1. Langages de skin, de style et de transformation

Les langages de skin, de style et de transformation n'ont pas tout à fait les mêmes objectifs.

Les langages de skin (peau en anglais) permettent de spécifier l'apparence graphique d'une IHM, c'est-à-dire l'IHM concrète. L'intérêt du langage de skin est de permettre à des non développeurs d'IHM de personnaliser leur IHM. Des logiciels comme le lecteur multimédia WinAMP (www.winamp.com), le logiciel de messagerie instantanée Trillian (http://www.ceruleanstudios.com) ou le navigateur FireFox (http://www.mozilla-europe.org/fr/products/firefox/) mettent en œuvre ce principe de skins. Il existe ainsi de nombreuses skins de ces logiciels regroupées sur des sites web. Les langages de skin sont des langages dédiés à la construction d'IHM concrètes. Ils peuvent être généralistes (cas du langage XUL pour FireFox) ou dédiés à un logiciel particulier (ex :WinAMP ou Trillian). Le niveau d'abstraction de ces langages est donc faible. Ils ne répondent pas à nos objectifs.

Les langages de style permettent de paramétrer l'apparence d'une IHM déjà conçue. Par paramétrage, on entend généralement les couleurs, fontes, bordures, marges, positionnement, etc. Il existe là encore de nombreux langages de style. Les langages de style peuvent être propres à une application, comme pour Word ou LaTeX. Ils peuvent aussi être publics comme [32 CSS] (Cascading Style Sheets). Ces langages ont tous vocation à modifier la « surface » des IHM. Il n'est pas question ici de modification de structure. Pour décrire les possibilités des langages de style, je retiens CSS. CSS est sans doute le langage de styles le plus utilisé pour le WEB, y compris par des non informaticiens.

Les langages de transformation permettent de modifier la structure même d'un document : création, suppression de parties, etc. Les langages de transformation ont pour but de transformer des structures en d'autres structures. Un cas classique d'utilisation en IHM est la concrétisation (par exemple, générer, à partir d'un modèle de tâches, une IHM en termes d'interacteurs). L'abstraction et la traduction (par exemple traduire du XUL vers du HTML) sont d'autres exemples. Bien entendu, les langages de transformation peuvent jouer le rôle de langages de style. J'étudie, dans cette section, trois langages de transformation : XSLT, lié au monde XML et au WEB ; les

transformations de graphes à base de grammaires de graphes, utilisées notamment par [70 Limbourg 2004] et enfin [6 ATL], issu du monde de l'IDM.

Que ce soient les langages de style ou les langages de transformation, le principe est toujours de sélectionner une partie d'un document (dans notre cas, le document représente un système interactif ou son IHM) et d'y apporter des modifications. Les documents peuvent être considérés comme des graphes : les nœuds sont les éléments du système interactif ; les arcs les relations entre ces éléments. Pour chaque langage de styles ou de transformation, j'étudie la puissance, d'une part, des sélecteurs offerts et, d'autre part, des transformations possibles. Le Tableau II-11 donne quelques exemples de transformation qu'il serait souhaitable de pouvoir exprimer. Aucun jugement de valeur n'y est fait. Les exemples illustrent simplement la puissance d'expression attendue.

Tableau II-11 : Exemples de transformations utiles.

Les entrelacements de haut niveau qui n'ont pas de descendant de type entrelacement sont rendus sous la forme d'un menu. Cette règle est classique en Web : l'entrelacement est rendu par une barre de navigation située à gauche de la fenêtre.
Les entrelacements de haut niveau qui ont des fils de type entrelacement sont rendus sous la forme d'un onglet. Les fils sont rendus sous la forme d'un menu. Cette règle est classique en Web : les onglets correspondent à la navigation de haut rang. Ils donnent accès à une navigation locale (à gauche) rendue sous la forme d'une barre de navigation.
Les choix simples à 2 éléments sont rendus sous forme d'interrupteurs. Les choix entre 3 et 7 éléments sont rendus sous forme de boutons radio. Les choix de cardinalité supérieure sont rendus sous forme de liste.
Les choix multiples qui ont plus de 10 éléments sont rendus sous forme d'accumulateurs.
Le conteneur direct d'une télécommande est masquable.
Les conteneurs placés dans une fenêtre ne peuvent pas être des fenêtres.
Supprimer tous les éléments d'un menu qui ne sont pas des liens hypertextes.
Remplacer tous les liens hypertexte d'un menu en boutons (c.f. section I de l'introduction, la maquette personnes âgées de la télécommande de Sedan-Bouillon).
Les conteneurs qui ne contiennent que des images sont masquables (publicité ?).

Pour l'analyse des langages, je me dote de deux grilles d'analyse : une pour les sélecteurs (Tableau II-12), l'autre pour les transformations proprement dites. Pour les sélecteurs, sont placées en abscisse :

- les opérations possibles : peut-on sélectionner un nœud par rapport à ses fils, ses pères, ses ancêtres ou ses descendants ?
- la possibilité de négation (ex: les fils qui ne sont pas…) ?
- les opérations de regroupement applicables aux résultats ? Union, intersection, différence ?

En ordonnée, sont examinés les types de résultats possibles. On distingue nœuds et sous graphes.

Pour les transformations proprement dites, six types de modification sont examinés :

- la modification de valeurs d'attributs (ex : couleur, fonte, etc.) ;

- la substitution d'un élément par un autre ;
- la substitution d'un élément par un groupe d'éléments ;
- la substitution d'un groupe d'éléments par un autre groupe ;
- la génération de contenu.
- la suppression de contenu.

Tableau II-12: Grille d'analyse des sélecteurs.

	Fils	Pères	Ancêtres	Descendants	Négation	Union	Intersection	Différence
Nœuds								
Sous graphes								

III.1.A. CSS : langage de style pour le WEB

[32 CSS] est utilisé pour définir les couleurs, les fontes, le rendu et d'autres caractéristiques liées à la présentation d'un document. L'objectif est de bien séparer la structure des documents WEB (HTML) de leur présentation. Le document HTML est vu comme un arbre dans lequel chaque balise est un nœud et chaque imbrication un arc. Par exemple, si un bouton est imbriqué dans un paragraphe, alors dans l'arbre le nœud correspondant au bouton est fils du nœud correspondant au paragraphe.

Le langage CSS a évolué au cours du temps. La version 3 est en cours de définition. Je me base ici principalement sur les spécifications de la version 2. Le langage CSS2 permet de définir un ensemble de règles de type <SELECTEUR, MODIFICATIONS> où SELECTEUR est une expression sélectionnant des nœuds du document WEB et MODIFICATIONS un ensemble de paramétrages à appliquer à ces nœuds (couleurs, fonte, bordure etc.). Ces règles peuvent être surchargés : c'est le principe de la cascade. Le document peut être accompagné de règles CSS2 mais l'utilisateur et/ou le navigateur peut rajouter ses propres règles CSS2 pour modifier le rendu du document WEB. Le Tableau II-13 donne quelques exemples de règles CSS2

Tableau II-13 : Exemples de règles CSS2.

H1, H2 {font-weight : bold ;}	Les titres de niveaux 1 et 2 sont en gras.
H1 > P:first-line { text-transform : uppercase;}	La première line des paragraphes fils d'un titre de niveau 1 est en majuscule.

Seuls les SELECTEURS sont ici étudiés. Les MODIFICATIONS serait fastidieuses à énumérer. Elles sont consultables dans [53 Hâkon 2005]. Disons simplement que CSS2 permet de modifier les attributs des balises HTML. Il est conçu pour la mise en forme de page WEB.

Un sélecteur CSS est une séquence de sélecteurs simples. Un sélecteur simple exprime les propriétés qu'un nœud doit satisfaire. Le Tableau II-14 illustre la syntaxe concrète de CSS2 et donne la sémantique associée.

Tableau II-14 : Résumé de la syntaxe et de la sémantique des sélecteurs CSS2.

Motif	Signification
*	Correspond à tout élément.
E	Correspond à tout élément E, c'est-à-dire de type E.
E F	Correspond à tout élément F qui est un descendant de l'élément E.

E > F	Correspond à tout élément F, enfant E.
E:first-child	Correspond à un élément E, premier enfant de son élément parent.
E:link E:visited	Correspond à un élément E de type ancre dont le lien n'a pas été visité (:link) ou l'a déjà été (:visited).
E:active E:hover E:focus	Correspond à l'élément E dans son interaction avec l'utilisateur. Il peut être actif (:active E), recevoir le focus (:focus) ou être survolé à la souris (:hover).
E:lang(l)	Correspond à l'élément de type E qui emploie une langue l (la détermination de cette langue est spécifique au langage du document).
E + F	Correspond à tout élément F immédiatement précédé par un élément E.
E[foo]	Correspond à tout élément E ayant l'attribut "foo" (quelles qu'en soient le nom et la valeur).
E[foo="foo-val"]	Correspond à tout élément E dont l'attribut "foo" a exactement la valeur "foo-val".
E[foo~="foo-val"]	Correspond à tout élément E dont l'attribut "foo" a pour valeur une liste de valeurs contenant "foo-val".
E[lang|="l"]	Correspond à tout élément E dont l'attribut "lang" a pour valeur une liste de valeurs commençant par "l".
DIV.foo-val	*Seulement en HTML*. Identique à DIV[class~="foo-val"].
E#myid	Correspond à tout élément E dont l'ID est "myid".
E, F	Correspond à tous les E et tous les F.

Les règles CSS2 sont simples à lire et à exprimer (cf exemples du Tableau II-13). Cependant, elles manquent de puissance à la fois dans les sélecteurs et les transformations. Du point de vue des sélecteurs, les manques sont les suivants :

- Il est impossible de sélectionner un nœud par la négation de son type. Par exemple, on ne peut pas sélectionner tous les descendants de E qui ne sont pas des F (ce qui aurait pu s'écrire E !F par exemple). Cette impossibilité sera levée avec CSS3 et la pseudo classe not. Cela donnera E *:not(F). La négation ne portera toutefois que sur le type d'un élément. On ne pourra pas sélectionner des nœuds qui n'ont pas de fils.
- Il est impossible de sélectionner des nœuds selon leurs descendants (seulement selon leurs ancêtres). Par exemple, on ne peut pas sélectionner les E qui ont parmi leurs descendants un F. Cette impossibilité sera très partiellement levée avec CSS3 et la pseudo class empty. E:empty permettra de sélectionner les E qui n'ont pas d'enfant.
- Il est impossible d'exprimer la différence ensembliste. Par exemple, on ne peut pas sélectionner les nœuds F qui ont E mais pas G comme ancêtre.
- Il est malaisé d'exprimer l'intersection ensembliste. Pour sélectionner les G qui ont à la fois un E et un F comme ancêtre, il faut écrire : E F G, F E G. Et encore cela n'est possible que si le graphe où s'applique ce sélecteur est un arbre.

Le Tableau II-15 place CSS2 dans la grille d'analyse des sélecteurs.

Tableau II-15 : Grille d'analyse des sélecteurs CSS2.

	Fils	Pères	Ancêtres	Descendants	Négation	Union	Intersection	Différence
Nœuds	∅	OUI	OUI	∅	Partiellement	OUI	Partiellement	∅

Sous graphes	∅	∅	∅	∅	∅	∅	∅	∅

Du point de vue des transformations proprement dites, seules les modifications d'attributs sont possibles.

III.1.B. XSL

[106 XSL] permet de transformer des documents de type XML. Il prend un document XML en entrée et produit un autre document. XSL est composé de trois langages : XPath, XSLT et XSL-FO. [107 XPath] est utilisé pour naviguer dans le document XML. [108 XSLT] permet de transformer le document. Enfin, XSL-FO permet de formater, de mettre en page le document en vue de le rendre sur papier, écran, etc. XSL-FO n'est pas examiné ici parce que ne répondant pas aux requis.

XPath : langage de sélecteurs

XPath considère le document XML comme un arbre. La navigation s'exprime par rapport à cette structure d'arbre. Une expression XPath caractéristique est un « chemin de localisation » constitué par une suite d'éléments ou d'attributs séparés par une barre de fraction (« / »). Les « chemins de localisation » sont divisés en étapes qui ont chacune 3 composants : un axe (AxisName), un nœud de test (NodeTest) et des prédicats (Predicate) (cf l'extrait de grammaire du Tableau II-16).

L'axe indique le type d'information qui sera sélectionné, relativement au nœud courant ou depuis la racine. Par exemple, *child::* est la syntaxe de l'axe des enfants du nœud courant. Dans beaucoup d'expressions XPath, quand l'axe n'est pas précisé, il s'agit implicitement de l'axe des enfants (child::). Un autre axe est par exemple celui des attributs.

Un nœud de test définit les éléments ou attributs à désigner. Le nœud de test le plus utilisé est le *test du nom* de l'élément ou d'un attribut.

Les prédicats sont des expressions plus complexes. Ils sont utilisés pour filtrer ou exclure certains nœuds. Les prédicats sont écrits entre crochets.

Tableau II-16 : Un extrait de la grammaire des expressions XPath.

Symbole	Traduction
LocationPath	RelativeLocationPath | AbsoluteLocationPath
AbsoluteLocationPath	'/' RelativeLocationPath?
RelativeLocationPath	Step | RelativeLocationPath '/' Step
Step	AxisSpecifier NodeTest Predicate*
AxisSpecifier	AxisName '::' | AbbreviatedAxisSpecifier
AxisName	'ancestor' | 'ancestor-or-self' | 'attribute' | 'child' | 'descendant' | 'descendant-or-self' | 'following' | 'following-sibling' | 'namespace' | 'parent' | 'preceding' | 'preceding-sibling' | 'self'
NodeTest	NameTest | NodeType '(' ')' | 'processing-instruction' '(' Literal ')'

La navigation dans le graphe s'appuie sur les axes. Le Tableau II-17 en donne la sémantique dans XPath1.

Tableau II-17 : Axes parcourables dans XPath1.

Axe	Signification
child axis	Donne accès aux enfants du noeud courant.

descendant axis	Donne accès aux descendants du noeud courant.
parent axis	Donne accès au père du noeud courant (le document est un arbre).
ancestor axis	Donne accès aux ancêtres du noeud courant.
following-sibling	Donne accès aux frères suivant le nœud courant.
preceding-sibling	Donne accès aux frères précédant le noeud courant.
following axis	Donne accès à tous les noeuds du document suivant le noeud courant sauf les descendants de ce noeud.
preceding axis	Donne accès à tous les noeuds du document précédant le noeud courant sauf les ancêtres de ce nœud.
attribute axis	Donne accès aux attributs de l'élément courant.
namespace axis	Donne accès aux namespaces du noeud courant.
self axis	Donne accès au noeud courant.
the descendant-or-self	Est équivalent à l'union de descendant et self.
ancestor-or-self axis	Est équivalent à l'union de ancestor et self .

XPath permet une grande liberté de navigation dans le graphe, notamment en donnant accès aux ancêtres d'un nœud. XPath permet de sélectionner des nœuds mais pas des sous graphes. XPath lève des limitations de CSS mais pas toutes. L'union et la différence ensemblistes peuvent être exprimées, mais difficilement et seulement dans le cas où le graphe est un arbre.

Tableau II-18 : Grille d'analyse des sélecteurs XPath1.

	Fils	Pères	Ancêtres	Descendants	Négation	Union	Intersection	Différence
Nœuds	OUI	OUI	OUI	OUI	OUI	OUI	Partiellement	Partiellement
Sous graphes	∅	∅	∅	∅	∅	∅	∅	∅

Notons que [107 Xpath2] est en cours d'élaboration. Il devrait surmonter toutes les limitations énoncées mais il sera plus verbeux. XPath2 prend en réalité l'aspect d'un mini langage de programmation.

XSLT : Langage de transformation

XSLT est la partie la plus importante de XSL. XSLT est utilisé pour transformer un document XML en un autre document XML. Classiquement, XSLT est utilisé pour transformer des documents XML en documents XHTML, de sorte à pouvoir les rendre sur un navigateur WEB. XSLT permet de générer, supprimer ou réarranger des balises ou des attributs du document XML original.

Les transformations en XSLT sont définies à l'aide de *templates*. Un template (balise <xsl:template match="XPATH">) identifie les éléments du document source à transformer (expression XPATH) et précise comment les réécrire dans le document cible. Un template peut faire appel à d'autres templates. Il est possible d'accéder aux éléments un par un (balise for-each), d'accéder aux sous arbres dont un élément est racine (balise value-of), de trier les éléments sélectionnés (balise sort) et de poser des conditions (balises if et choose).

Des travaux montrant comment utiliser XSLT pour les transformations de modèles en IHM ont été menés [75 Martínez-Ruiz 2006].

La Figure II-31 donne un exemple de règle de transformation XSLT. Le document source est un modèle de tâches au format XML. La règle précise qu'on recherche les

entrelacements de plus haut niveau (ceux qui ne sont pas descendants d'un autre entrelacement). Chaque fils de ces entrelacements (c'est-à-dire chaque tâche entrelacée) est affiché dans un paragraphe. Un séparateur (<hr/>) est ajouté après chaque paragraphe.

```
<xsl:template match="/">  // Ce template correspond à la racine
  <html>  // Code HTML généré
    <body>
     // Pour chaque entrelacement qui ne contient pas d'entrelacement
     // appliquer les templates.
      <xsl:apply-templates select="Interleaving[not(./ancestor::Interleaving)]"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="Interleaving">  // Si on a un entrelacement
  <xsl:for-each select="./*">  // Pour chacun des fils…
    <p><xsl:value-of select="."></p>  // Placer le contenu du fils dans un paragraphe
    <hr/>  // Ajouter un séparateur
  </xsl:for-each>
</xsl:template>
```

Figure II-31 : Exemple simplifié de transformation XSLT d'un modèle de tâches en un modèle HTML.

XSL est un langage puissant. Il permet de modifier tous les éléments du document source. Il est possible de générer ou supprimer du code. La seule contrainte est de disposer d'un document au format XML. Dans le cadre de la plasticité, l'utilisation typique de XSLT serait, à partir d'un modèle d'IHM (niveau tâche, AUI ou CUI), de concrétiser ce modèle en utilisant un document XSLT approprié au contexte d'usage.

III.1.C. Grammaires de graphes

Les grammaires de graphes ont notamment été utilisées en IHM par [70 Limbourg 2004]. Le principe est de disposer d'un graphe modélisant l'IHM. A partir de ce graphe, il est possible d'appliquer une série de transformations pour obtenir un nouveau graphe. Le principe est proche de celui mis en œuvre dans XSLT : on part d'un document pour arriver à un autre document. L'algorithme de transformation recherche dans le graphe les sous graphes sur lesquels appliquer les règles de transformations. Les transformations sont définies en quatre parties.

- PAC (Positive Application Condition) : fixe les conditions que le sous graphe doit remplir pour que la transformation puisse s'opérer.
- NAC (Negative Application Condition) : fixe les conditions que le sous graphe ne doit pas remplir pour que la transformation puisse s'opérer.
- LHS (Left Hand Side) : définit le graphe « motif » à transformer.
- RHS (Right Hand Side) : définit la transformation à appliquer.

Le mécanisme de sélection est défini par les parties PAC, NAC et LHS. Le sous graphe décrit dans LHS est sélectionné pourvu que PAC soit vérifié sur ce sous graphe et que NAC ne le soit pas. Sur l'exemple de la Figure II-32, tous les joueurs des Louvain United (LHS) ayant un salaire supérieur à 3000 (PAC) sont sélectionnés pour le match

du 4 juin 2004 (RHS). La partie RHS indique comment LHS doit être réécrit dans le document résultat. Sur l'exemple, les joueurs sélectionnés sont assignés au match du 4 juin 2004 (RHS). RHS peut créer des nœuds, en supprimer, modifier des attributs, des relations, etc.

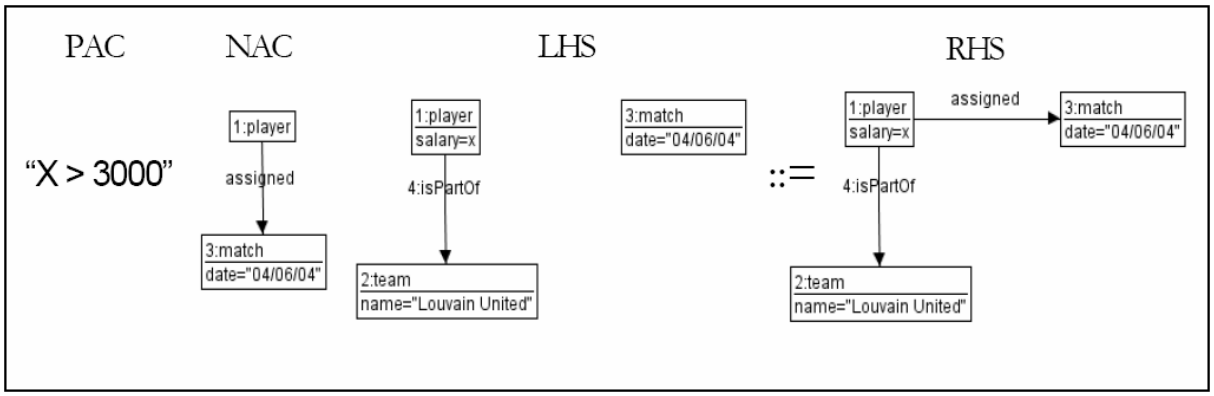


Figure II-32 : Exemple de règle de transformation de graphe. Extrait de [70 Limbourg 2004].

Contrairement aux solutions précédentes (CSS et XPath), il est ici possible de sélectionner des sous graphes (pas seulement des nœuds). Les sous graphes sont ensuite réécrits. C'est là que des informations peuvent être ajoutées, modifiées ou supprimées.

Il ne semble, par contre, pas possible de sélectionner des sous graphes en fonction des relations qu'ils auraient avec des nœuds non directement liés à ces sous graphes. Par exemple, il n'est pas possible de parler des nœuds « petits-enfants » d'un autre nœud sans parler du noeud correspondant. En particulier, il ne semble pas possible de sélectionner un nœud par rapport à ses ancêtres ou ses descendants. Ceci est une limite forte des sélecteurs. De même, s'il est possible d'exprimer l'intersection (PAC et LHS) et la différence (PAC, NAC, LHS), il ne semble pas possible d'exprimer directement l'union (ex : les joueurs qui ont joué un match ou qui font partie de telle équipe). Le Tableau II-19 résume ces points.

Tableau II-19 : Grille d'analyse des sélecteurs dans les grammaires de graphes.

	Fils	Pères	Ancêtres	Descendants	Négation	Union	Intersection	Différence
Nœuds	OUI	OUI	Partiellement	Partiellement	OUI	NON	OUI	OUI
Sous graphes	OUI	OUI	Partiellement	Partiellement	OUI	NON	OUI	OUI

III.1.D. ATL : langage de transformation généraliste

[6 ATL] vient de la communauté IDM. C'est un langage de transformation de modèles. Un programme ATL est composé de règles qui définissent comment les éléments du modèle source sont parcourus et transformés pour produire le modèle cible. ATL permet de modifier la structure d'un document. Il est possible de créer et supprimer des éléments, de modifier des attributs, etc. ATL n'a pas à proprement parlé de mécanisme de sélection. ATL est un mini langage de programmation. Il est possible d'implémenter tous les mécanismes de sélection en ATL, mais aucun sous langage dédié, semblable par exemple à CSS ou XPath, n'est fourni de base. Je montre dans cette section comment il est possible d'implémenter des sélecteurs à la CSS ou à la XPath en ATL. Pour cela, il faut définir, en premier lieu, un métamodèle de documents (Figure II-33). Les règles ATL sont basées sur ce métamodèle.

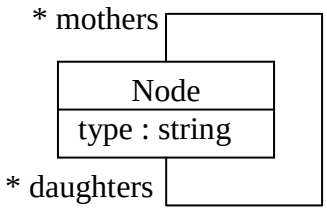


Figure II-33 : Metamodèle minimaliste d'un document de type graphe.

A titre d'illustration, je donne la fonction *Desc* (pour Descendants) qui prend en paramètre un ensemble de nœuds *S_n* et un type *t*. Elle renvoie un ensemble de nœuds : ceux de type *t* qui sont descendants d'un des nœuds de *S_n*.

```
Helper def : Desc(set {Node} : S_n, string : t) : set {Node} =
// Soit res l'ensemble résultat
  let res = set {} in
// Pour chaque noeud de S_n
    for n in S_n {
// Si n est de type t
      if (n.type = t) then {
// Ajouter à res n et le résultat de la récursion sur les fils de n
  res.union(n.union(for e in n.daughters {Desc(e.daughters, t)})
// Sinon ajouter à res le résultat de la récursion
 else {for e in n.daughters {res.union(Desc(e.daughters, t))} }
endif ;
```

Supposons maintenant disponibles les fonctions suivantes :

- Nodes(string : t) : renvoie les nœuds de type t.
- Child(set {Node} : S_n, string t) : renvoie les nœuds de type t qui sont fils d'un des nœuds de S_n.
- Nodes_have_desc(set {Node} : S_n, set {Node} : S_d) : renvoie les noeuds de S_n qui ont parmi leurs descendants au moins un noeud de S_d.

On peut alors donner quelques exemples de « sélecteurs » ATL :

- « E F » en CSS devient Desc(Nodes("E"), "F").
- « E F > G » en CSS devient Child(Desc(Nodes("E"), "F")).
- Les E ayant des F pour descendants se traduit par : Nodes_have_desc(Nodes("E"), Nodes("F")).

Il semble relativement aisé de proposer un ensemble de fonctions implémentant les bases des sélecteurs CSS ou XPath, voire d'étendre ces bases. Cela pourrait être fait par le concepteur pour peu qu'il maîtrise ATL. Cependant, pour qu'ATL soit opérationnel, il faut au préalable charger le document dans la machine ATL. Ceci risque d'alourdir le processus de transformation du document.

La critique déterminante qui peut être faite à ATL est qu'il n'aide pas plus qu'un autre langage à définir des sélecteurs à la CSS ou XPath. Les fonctions données à titre d'exemple pourraient facilement être implémentées dans tout langage de programmation (ou presque). ATL a été conçu pour de la transformation entre métamodèles. C'est un langage généraliste. Il ne présuppose aucune structure aux documents : ils doivent juste être conformes à leurs métamodèles. XSLT, par exemple, a

été conçu pour être appliqué à des documents XML. Ceux-ci sont structurés en arbres. Ainsi, indépendamment de la sémantique du document, il est possible de la parcourir (grâce à XPath). ATL ne veut rien supposer de tel, ce qui explique l'absence d'un langage de sélecteur plus évolué.

III.1.E. Conclusion sur les langages de style et de transformation

Les langages de style comme CSS offrent simplement des adaptations de surface (modification d'attributs). Les langages de transformation permettent de modifier les documents en ajoutant, supprimant ou remplaçant des informations. Dans le cas des langages de transformation basés sur des documents dont la structuration est connue (ex : XML), il est possible d'employer des langages dédiés de sélecteurs s'appuyant sur cette structuration (ex : la relation père-fils en XML).

Le Tableau II-20 compare informellement CSS, XPath et les grammaires. De cette comparaison informelle, il apparaît assez clairement que XPath est le langage de sélecteurs le plus puissant. Cependant, les expressions de différence ou d'intersection sont verbeuses et limitées à des structures d'arbre.

Tableau II-20: Comparaison informelle des sélecteurs CSS, XPath1 et des grammaires de graphe.

Objectif	CSS 2-3	XPath 1	Grammaires
Les fils de A qui ne sont pas des B	A>*:not(B) (CSS3)	A/*[name() !='B']	PAC : A→X≠B
Les fils de A qui ne sont ni des B ni des C	∅	A/*[name() !='B' and name() != 'C']	PAC : A→X Avec X≠B et X≠C
Les B fils de A	A > B	A/B	A→B
Les B descendants de A	A B	A//B	∅
Les A qui ont des B pour descendants	∅	B/ancestor::A	∅
Les B qui n'ont pas de A pour ancêtres	∅	B[not(ancestor::A)]	∅
Les ancêtres de A	∅	A/ancestor::*	∅
Les A qui n'ont pas de B pour fils	∅	A[not(B)]	NAC : A→B LHS : A
Les A qui n'ont que des B pour fils	∅	A/not(*[name() != 'B'])	NAC : A→X≠B
Les C qui ont au moins un A et un B pour ancêtre	A B C, B A C (Valide seulement sur un arbre.)	C[ancestor::A][ancestor::B]	∅
Tous les B sauf ceux qui sont fils d'un A	∅	B[not(./parent::B/parent::A)] (Valide seulement sur un arbre.)	NAC : A→B LHS : B
Les A qui ont plus de 4 fils	∅	A[number(*) > 4]	PAC : A→B A→C A→D

			A→E A→F LHS : A
Les A qui ont plus de B que de C pour fils	∅	A[count(B) > count(C)]	∅
Les E qui ont un C ou un D pour père ; C et D ayant un A ou un B pour ancêtre	A C>E, A D>E, B C>E, B D>E	A///C/E | A///D/E | B//C/E | B//D/E ou *[name()='A' or name() = 'B']//*[name()='C' or name()='D']/E	∅
Les A plus proches ancêtres d'un B	∅	B/ancestor::A[0] (Valide seulement si la liste des ancêtres est classée selon le nombre de génération.)	∅
Les A plus lointains ancêtres d'un B	∅	A[not(./parent::A/ancestor::A)][.//B] (Valide seulement sur un arbre.)	∅
Les A qui ont une majorité absolue de B pour fils	∅	A[count(B) > count(*[name() != 'B'])]	∅
Les B descendants de A qui ne sont pas séparés de ce A par un C	∅	∅	∅

III.2. Les annuaires dans les approches orientées services

Les approches orientées services, par la notion d'annuaire, peuvent apporter une réponse à l'imprévisibilité du contexte d'usage. La philosophie de ces approches est de construire des applications par assemblage de services. Un service peut être vu comme une boîte noire offrant certaines fonctionnalités. Ces fonctionnalités et la façon d'y accéder sont décrites à l'aide d'un langage. Une architecture orientée service (SOA pour Software Oriented Architecture) fonctionne en quatre étapes. Premièrement, des fournisseurs enregistrent leurs services dans un ou plusieurs annuaires. Deuxièmement, des clients interrogent ces annuaires pour y trouver des services selon une description qu'ils donnent. Troisièmement, l'annuaire renvoie les accès des services correspondant aux descriptions fournies. Enfin, le client utilise les services trouvés (Figure II-34).

Je me focalise ici sur les annuaires, c'est-à-dire le moyen de classer et de retrouver des services.

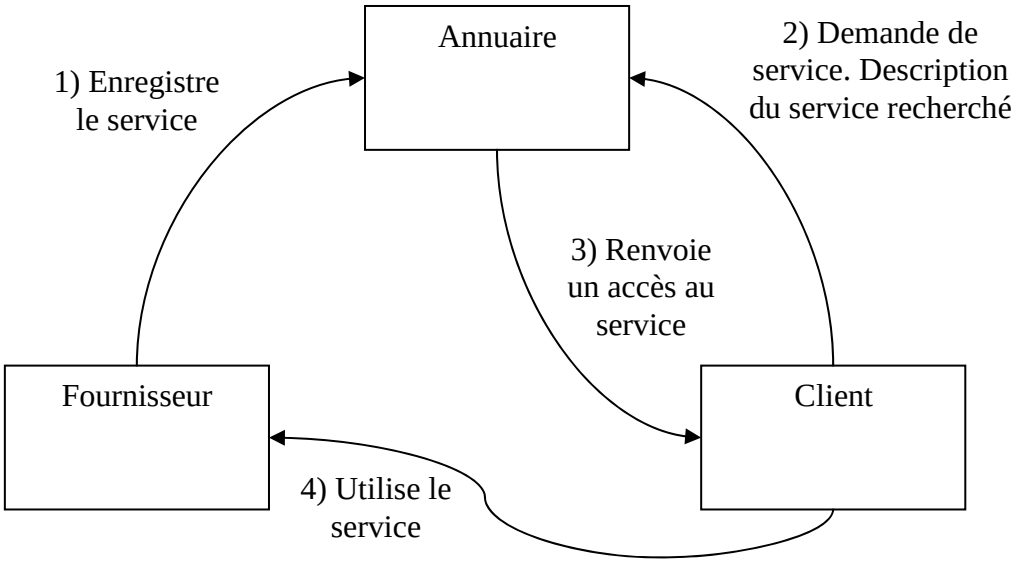


Figure II-34 : Principe d'exploitation des architectures orientées services.

Historiquement, les SOA se sont intéressés à modéliser la partie fonctionnelle des services. Il s'agissait de pouvoir faire coopérer des systèmes préexistants. Les SOA ont réellement commencé à avoir du succès avec les Services WEB. Les Services WEB ont, de fait, apporté des standards : [105 WSDL] (Web Service Description Language) comme langage de description de service et [100 UDDI] (Universal Description Discovery and Integration) comme norme d'annuaire de service. Ces standards sont étudiés dans la première partie.

D'autres travaux sur les SOA proviennent du Web Sémantique. Le Web Sémantique vise à rendre le contenu du WEB accessible et utilisable par des programmes et non plus seulement des humains. L'idée est d'associer des méta-données dans un langage formel, par exemple [81 OWL] (Web Ontology Language). Avec le développement des Web Services, l'idée est de décrire des services de la même manière que sont décris les documents. Ainsi, le langage [82 OWL-S] offre une ontologie dédiée à la description des services. Le projet [3 Amigo] se base sur OWL-S pour définir son propre langage de description de service : SD-SDCAE (Service Description – Service Discovery, Composition, Adaptation & Execution). L'étude de OWL-S et SD-SDCAE est l'objet de la deuxième partie.

Des approches associent à la modélisation sémantique une modélisation de l'IHM [85 Ponnekanti 2001] [62 Khushraj 2005] [104 Vermeulen 2007]. Cependant, ces travaux ne disent pas comment retrouver les services d'un point de vue de l'IHM. Je ne les examine donc pas.

III.2.A. UDDI (Universal Description Discovery and Integration) et WSDL (Web Service Description Language)

[100 UDDI] est une norme d'annuaire de services. [105 WSDL] est un langage de description technique de services. Tous deux sont issus du monde Web et représentent un standard de facto pour les entreprises. UDDI est à la fois un modèle de données permettant de décrire un service Web et la définition d'une interface permettant de manipuler ce modèle de données. UDDI est dérivé de LDAP (Lightweight Directory Access Protocol, [63 LDAP]). Alors que le modèle de données LDAP est extrêmement générique et que chaque système d'information peut définir son schéma LDAP, le

modèle de données UDDI est un modèle plus figé. Ainsi, s'il est possible de représenter et stocker un annuaire UDDI dans un annuaire LDAP (après avoir pris soin de définir un schéma UDDI pour LDAP), la réciproque n'est pas vraie dans le cas général. Le modèle de données UDDI est défini sous la forme de schémas W3C XML Schéma *(http://www.uddi.org/schema/uddi_v2.xsd).* La spécification UDDI définit explicitement quelles sont les entités qui se trouvent dans l'annuaire (Figure II-35) :

- Les « businessEntities » sont en quelque sorte les pages blanches d'un annuaire UDDI. Elles décrivent les organisations ayant publié des services dans le répertoire. On y trouvera notamment le nom de l'organisation, ses adresses (physiques et Web), des éléments de classification, une liste de contacts. Chaque businessEntity est identifiée par une « businessKey ».
- Les « publisher assertions » sont des moyens d'associer des businessEntities. Leur raison d'être est que beaucoup d'entreprises ne sont pas correctement représentées par une seule businessEntity. Par exemple, une grande entreprise peut avoir de nombreuses filiales qui sont elles-mêmes décrites. Cette même grande entreprise peut vouloir expliciter le lien qu'elle entretient avec ses filiales dans l'annuaire UDDI.
- Les « serviceEntities » sont en quelque sorte les pages jaunes d'un annuaire UDDI. Ils décrivent de manière non technique les services proposés par les différentes organisations. On y trouvera essentiellement le nom et la description textuelle des services ainsi qu'une référence à l'organisation proposant le service et un ou plusieurs « bindingTemplates ».
- Les « bindingTemplates » donnent les coordonnées des services. Cherchant à être très générique en ce domaine, UDDI permet de décrire des Services Web HTTP, mais également des services invoqués par d'autres moyens (SMTP, FTP, fax, téléphone, ...). Ils contiennent notamment la définition du « point d'accès » (suivant les cas, une URL, un numéro de téléphone, ...) et les éventuels « tModels » associés.
- Les « tModels » sont les descriptions techniques des services. UDDI n'impose aucun format pour ces descriptions qui peuvent être publiées sous n'importe quelle forme et notamment en documents textuels (XHTML par exemple). C'est à ce niveau que WSDL intervient comme le vocabulaire de choix pour publier des descriptions techniques de services.

La Figure II-35 résume les relations existant entre les différentes entités présentes dans UDDI.

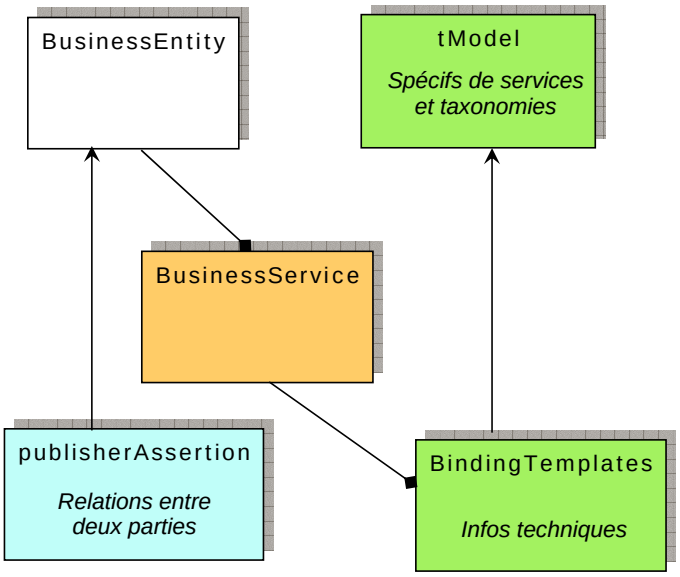


Figure II-35 : Les principales entités dans UDDI.

Les annuaires UDDI disposent d'une interface réseau pour traiter les requêtes des clients cherchant des services Web. Il est possible de faire des recherches portant sur les BusinessEntities, les BusinessServices, les BindingTemplates et les tModels. La Figure II-36 illustre le langage utilisable dans UDDI pour faire une requête portant sur un tModel.

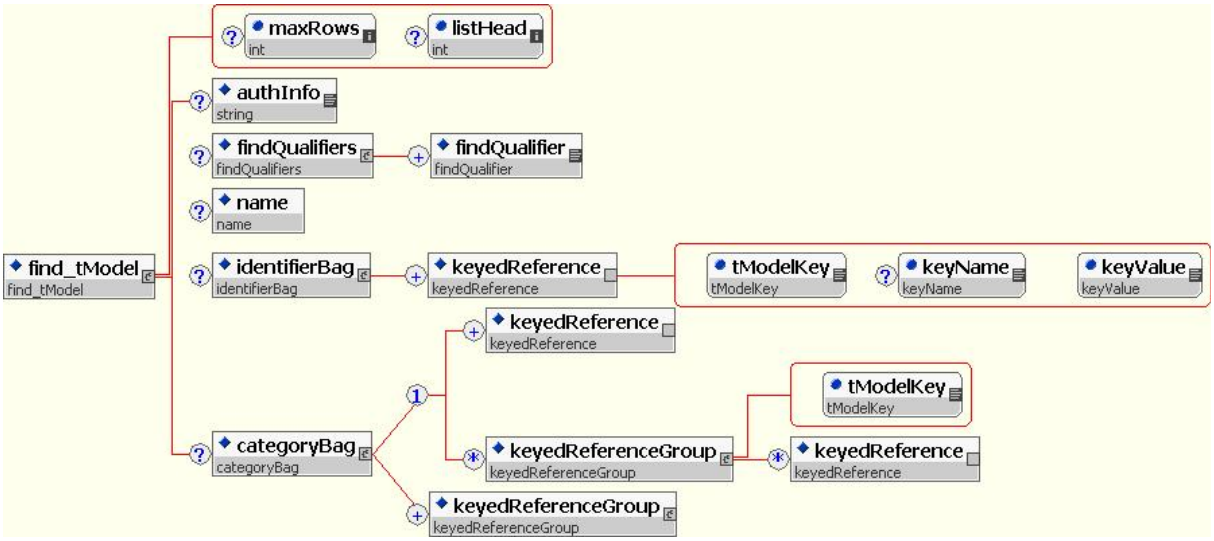


Figure II-36 : Description du langage de requêtes UDDI pour les tModels.

Les requêtes que l'on peut poser à UDDI souffrent d'une limitation importante. On ne peut techniquement pas chercher des services répondant à une description WSDL. Il faut procéder en deux temps : d'abord faire une recherche par mot clefs pour trouver les services susceptibles de correspondre à la description cherchée (par exemple un service de réservation de billets) puis, hors UDDI, analyser les descriptions WSDL pour vérifier leur correspondance avec la requête initiale.

La section suivante vise à décrire brièvement WSDL et à en montrer les limites.

WSDL (Web Service Description Language)

Un document WSDL décrit de façon technique un service Web. Il est principalement composé de sept éléments :

- définitions est l'élément racine du document. Il donne le nom du service, déclare les espaces de noms utilisés et contient les éléments du service.
- message décrit les données transmises. Chaque message se compose d'une ou plusieurs parties. Ces parties sont l'équivalent des paramètres d'un appel de fonction dans les langages de programmation.
- portType décrit un service, les opérations qui peuvent être réalisées et les messages impliqués dans ces opérations. Le portType est l'équivalent d'une bibliothèque de fonctions dans les langages de programmation.
- types décrit tous les types de données utilisés entre le client et le serveur. WSDL n'est pas exclusivement lié à un système spécifique de typage, mais utilise par défaut la spécification XML Schema.
- binding décrit les spécifications concrètes de la manière dont le service sera implémenté, c'est-à-dire le protocole de communication et format des données pour les opérations et messages définis par un type de port particulier. Généralement les binding se font vers SOAP, mais WSDL n'est pas restreint à SOAP.
- service définit les adresses permettant d'invoquer le service donné. Ceci sert à regrouper un ensemble de ports reliés. La plupart du temps, c'est une URL invoquant un service SOAP.
- documentation contient des documents en langue naturelle.

En termes informatiques, WSDL fournit l'équivalent du fichier .h en C++. Les seuls raisonnements qu'il permet de tenir en recherche de services Web sont de niveau syntaxique : puis-je envoyer tel message sur tel port ? Comment le formater ? etc. Tout raisonnement de niveau sémantique (que fait cette opération ?) ne peut être que semi-automatique, aidé par un humain.

Conclusion

UDDI est la première tentative conséquente pour capitaliser les services Web dans un annuaire. A l'origine, le but essentiel était l'intégration et la semi-automatisation des échanges intra et inter entreprises. UDDI est devenu un standard dans le monde industriel. Ce standard est lié à WSDL, utilisé comme langage de description technique des services Web répertoriés dans UDDI.

Critique

Le duo UDDI + WSDL ne remplit pas la mission de fournir un annuaire réellement exploitable automatiquement. Il n'est pas possible de poser des requêtes UDDI pour chercher un service Web à partir d'une description WSDL. Il faut d'abord rechercher les services Web à l'aide de mots clefs, en ciblant éventuellement des entreprises particulières. C'est ensuite seulement qu'un algorithme de reconnaissance peut être appliqué aux descriptions WSDL trouvées.

Notons de plus que même si de telles requêtes étaient possibles, WSDL n'exprimerait que des informations syntaxiques sur le service. Ainsi, il ne semble pas réellement possible de poser des requêtes d'ordre sémantique telles que « chercher un service qui fasse des réservations de billets ».

III.2.B. Amigo SD-SDCAE

Les travaux sur SD-SDCAE (Service Description – Service Discovery, Composition, Adaptation & Execution) ont été menés dans le cadre du projet [3 Amigo]. Le projet Amigo s'intéresse à l'informatique ambiante. Il s'agit de proposer des services capables de répondre aux besoins de l'utilisateur dans un environnement dynamique. En particulier, les plates-formes se découvrent dynamiquement. Pour cela, Amigo adopte le principe d'une architecture à services offrant des descriptions syntaxiques et sémantiques.

Si l'aspect syntaxique de telles descriptions est depuis longtemps résolu (WSDL par exemple), il n'en va pas de même pour l'aspect sémantique. L'enjeu est de savoir non seulement si deux services **peuvent** communiquer syntaxiquement mais aussi et surtout si **cela à un sens** sémantiquement.

L'approche retenue dans Amigo pour exprimer la sémantique s'appuie sur OWL-S. OWL (Ontology Web Language) est une ontologie pour le Web, développée dans le cadre des recherches sur le Web sémantique. OWL-S s'en inspire pour proposer une ontologie des services (Délivrable D3.1). Ces ontologies sont utilisées pour déterminer de façon automatique si deux services sont similaires, c'est-à-dire s'ils peuvent fournir des fonctionnalités similaires et garantir des propriétés similaires.

Un service SD-SDCAE est décrit en plusieurs parties :

- ServiceProfile exprime ce que fait le service. Son rôle est de faire la publicité à l'extérieur des capacités du service. On y retrouve décrites les entrées et sorties du service, les pré et post conditions. On retrouve aussi la catégorie du service exprimée dans une taxonomie donnée. Le serviceProfile est l'équivalent de la notion d'interface publique en JAVA. (Figure II-37).

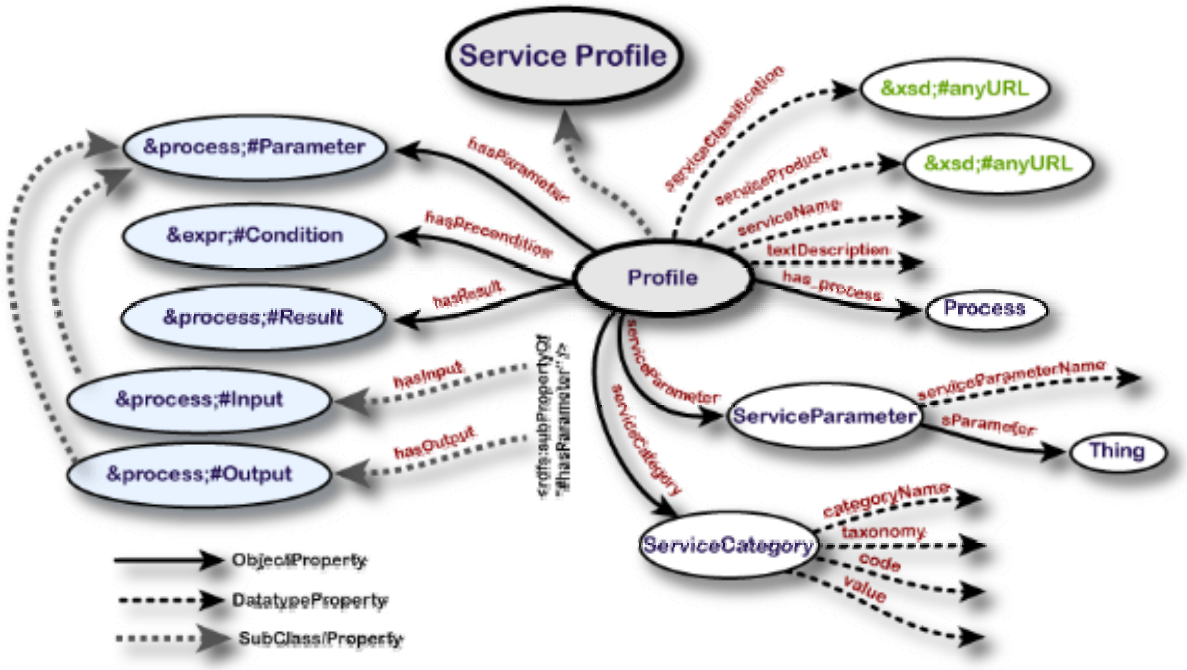


Figure II-37 : Le ServiceProfile décrit publiquement le service.

- ServiceModel exprime comment fonctionne le service. C'est un modèle de dialogue orienté service. Il a pour but de pouvoir faire automatiquement dialoguer des services entre eux. Comme le service profile, ce modèle définit

des entrées/sorties, pré conditions et effets. Ces informations devraient être compatibles avec celles exprimées dans le service profile mais rien ne l'oblige formellement. Le service model définit la structure des échanges. Il précise une notion d'interface privée : en effet, ces informations ne seront pas forcément rendues publiques. Lors de la sélection d'un service par requête, le serviceModel permet d'élaguer les réponses possibles. On pourra, par exemple, exclure un service de reconnaissance de doigts qui impose au préalable une phase de calibrage. On privilégiera ceux dont le calibrage est automatique, inexistant ou déjà réalisé. (Figure II-38)

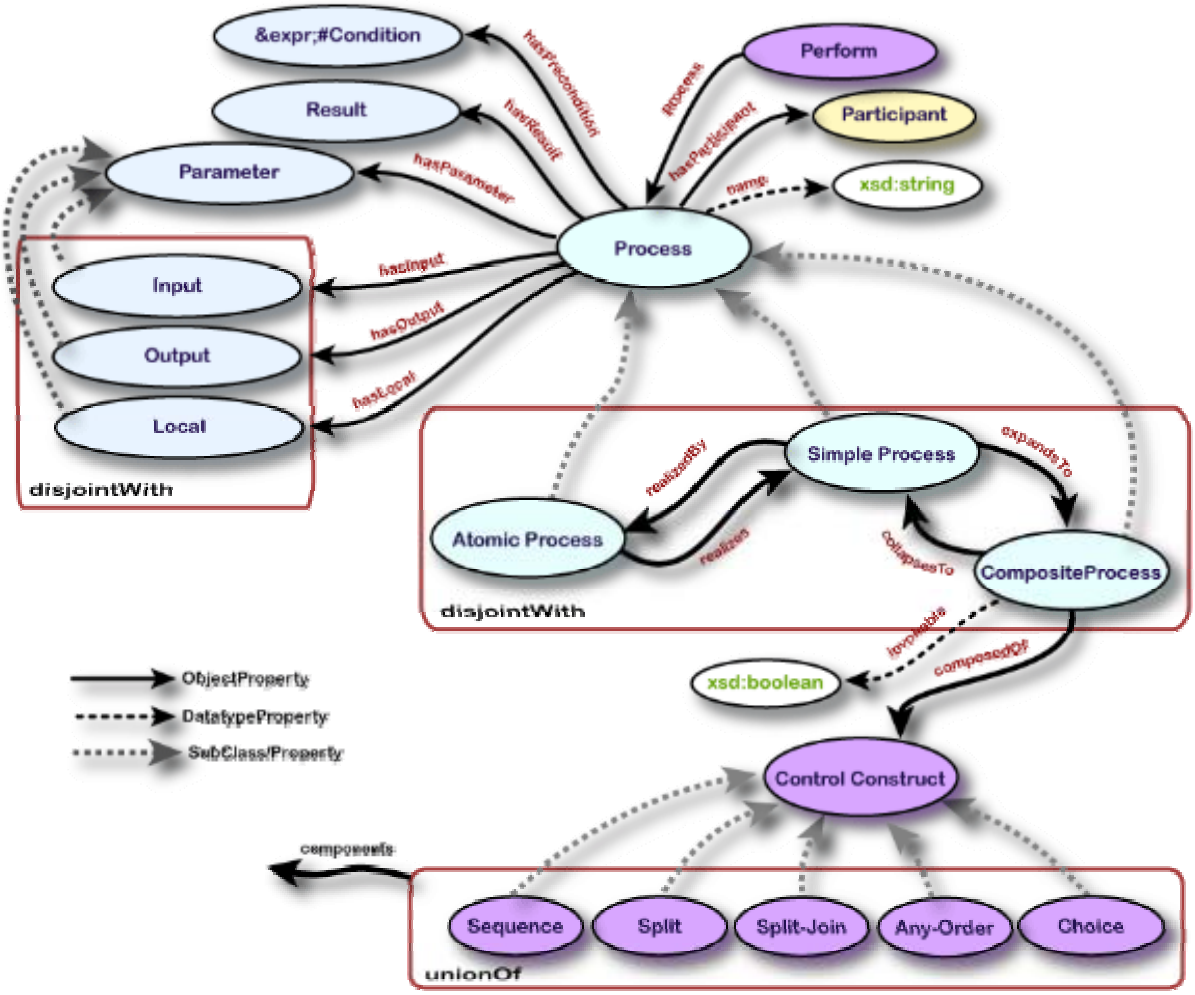


Figure II-38 : Le ServiceModel décrit le fonctionnement interne du service.

- ServiceGrounding exprime la manière d'accéder au service. On y retrouve les informations relatives aux protocoles mis en œuvre (SOAP, http, etc.) et les liens avec les modèles précédents. Sans détailler cette partie, je dirais simplement qu'elle se rapproche des bindings WSDL. Il s'agit de décrire comment concrètement le service peut communiquer.

La Figure II-39 illustre l'architecture prenant en charge la recherche de services dans SD-SDCAE. Les fournisseurs commencent par enregistrer leurs services dans l'annuaire (1). Ces services sont classés en utilisant un algorithme de matchmaking (2). Cet algorithme reconnaît dans quelle mesure deux services sont proches sémantiquement. Il est utilisé dans l'annuaire pour classer les services entre eux. Une fois les services enregistrés, le client fait une demande en donnant une description du service qu'il

recherche (3). L'architecture recherche dans l'annuaire de services ceux correspondant au service demandé (4). Il utilise pour cela un algorithme de matchmaking. Les résultats sont filtrés en fonction de l'adéquation des services par rapport au contexte et à la qualité de service demandée (5-6). Le service trouvé est intégré chez le client (7). La tâche est éventuellement adaptée (8) et enfin rendue disponible.

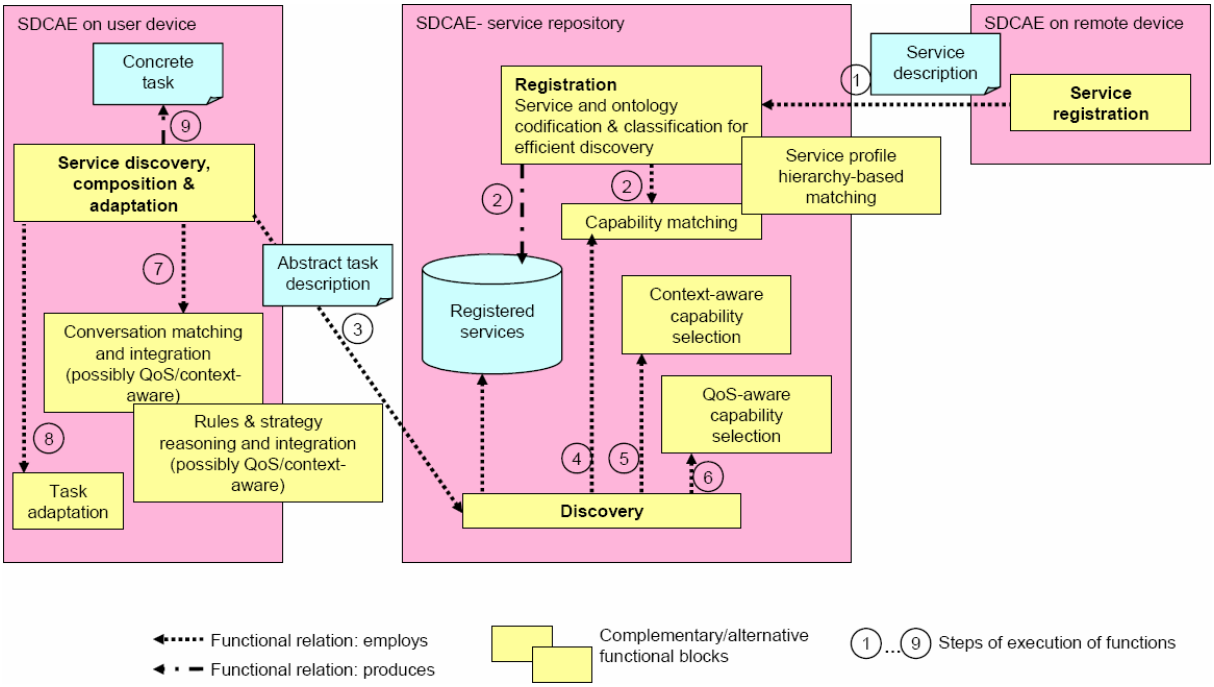


Figure II-39 : Fonctionnement général de la recherche de service. (Extrait de Amigo D3.3)

J'examine ci-dessous les algorithmes de matchmaking.

Les algorithmes de matchmaking

La recherche de services se fait en posant une requête décrivant le service recherché. Un algorithme de reconnaissance prenant en paramètres la description du service recherché et la description d'un service proposé est utilisé pour fournir au client les services désirés : c'est ce type d'algorithme qu'on qualifie de matchmaking. L'algorithme de matchmaking basique utilisé dans les approches à base d'ontologie provient de [83 Paolucci 2002]. Il se base sur une modélisation des services en termes d'entrées et de sorties. Son principe est le suivant : Il y a correspondance entre le service proposé et le service recherché lorsque d'une part, toutes les sorties du service recherché se retrouvent dans les sorties du service proposé et, d'autre part, toutes les entrées du service proposé se retrouvent dans les entrées du service recherché. Cela garantit que le service proposé fournit les informations demandées et que le client peut lui fournir ce dont il a besoin pour fonctionner. La correspondance entre un service proposé et un service demandé dépend donc de la correspondance des entrées et sorties de ces deux services.

La correspondance entre ce qui est requis (R) et ce qui est fourni (F) en entrée ou en sortie n'est pas binaire. Elle peut prendre quatre valeurs :

- Exact : Si R = F ou si R sous type F, en faisant l'hypothèse que F agit de façon cohérente avec ses sous types.
- Plug in : Si F subsume R, c'est-à-dire que F est plus général que R.

- Subsume : Si R subsume F, c'est-à-dire que F est plus spécialisé que R. Dans ce cas, F est utilisable mais probablement insuffisant.
- Fail : Si aucunes des relations précédentes n'a pu être établie.

Dans le cas où plusieurs services sont reconnus, [83 Paolucci 2002] propose un algorithme pour les classer. La priorité est donnée aux sorties :

sortRule (F1, F2 : deux descriptions de services fournis. R : une description du service requis) {
Si les sorties de F1 correspondent mieux à R que F2, alors F1 > F2
Si les sorties de F1 et F2 sont équivalentes pour R et si les entrées de F1 correspondent mieux à R que F2, alors F1 > F2
Si F1 et F2 ne se distinguent pas par rapport à R, alors F1 = F2
}

SD-SDCAE se base sur ces travaux mais affine le calcul de la similitude. Le calcul de la similitude entre deux services est basé sur la propriété de subsomption sur l'ontologie, c'est-à-dire la propriété de généralité : A subsume B si A est équivalent à B ou A inclut B. Autrement dit, A peut être utilisé partout ou B peut l'être. Informellement, SD-SDCAE définit la notion de distance entre deux concepts comme le nombre de concepts qui les séparent dans l'ontologie. En réalité, cette distance n'en est pas vraiment une. En effet, on note dans la définition ci-dessous que d(A,B) n'est pas égal à d(B, A).

$$d(A,B)=\{si \neg Subsume(A,B) \text{ alors NULL sinon } \#\{C \mid Subsume(A,C) \wedge Subsume(C,B)\}\}$$

A partir de cette notion de distance, SD-SDCAE propose une formule de reconnaissance d'un service (C2) par rapport à une description de service donnée (C1). Cette formule exprime que :

- Toutes les entrées de C2 sont présentes dans C1 (1ère ligne) ;
- Toutes les sorties recherchées (C1) sont offertes par C2 (2ème ligne) ;
- Toutes les propriétés attendues (C1) sont offertes par C2 (3ème ligne).

$$\begin{aligned} Match(C1, C2) = {} & \forall in' \in C1.In, \exists in \in C2.In : d(in, in') \geq 0 \\ & \wedge \forall out' \in C2.Out, \exists out \in C1.Out : d(out, out') \geq 0 \\ & \wedge \forall p' \in C2.P, \exists p \in C1.P : d(p, p') \geq 0 \end{aligned}$$

Le classement des services correspondant à une requête se fait en utilisant la distance sémantique des services trouvés par rapport au service demandé. La distance entre deux services et calculé comme étant la somme des distance entre les entrées, les sorties et les propriétés.

La Figure II-40 illustre l'application de cet algorithme sur un service de streaming vidéo. Le service de streaming digital est à une distance de 3 de ce service (2 entre digital ressource par rapport à Video Ressource et 1 entre Digital Server et Video Server).

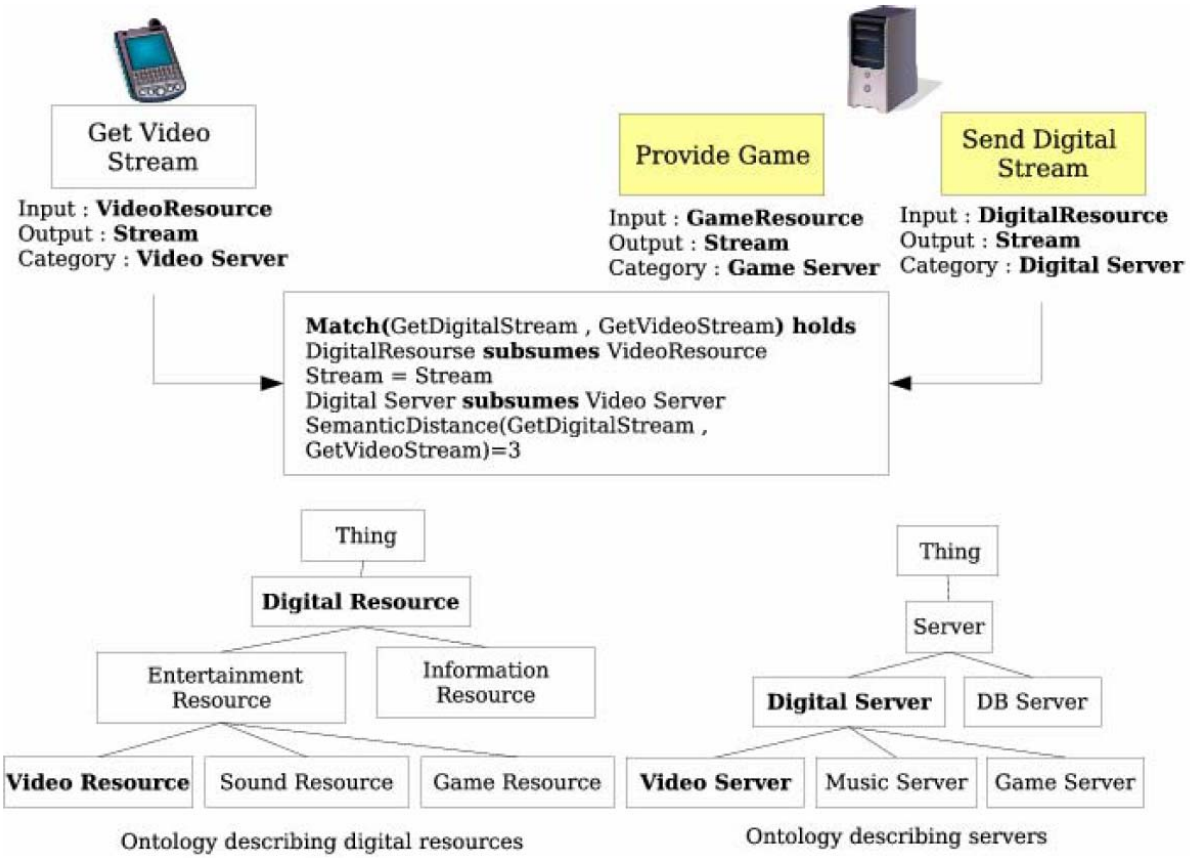


Figure II-40 : Exemple de recherche d'un service de streaming video.

Matchmaking : le problème des performances

Le principal point faible de ces approches est le temps de calcul nécessaire. SD-SDCAE propose une méthode pour accélérer la recherche de services et l'ajout de nouveaux services. Le principe est un pré classement des services. Les services sont placés dans un graphe où chaque arc représente la relation de correspondance (matching) (Figure II-41). Le graphe est composé d'îlots, chacun correspondant à un ensemble de services utilisant des ontologies communes.

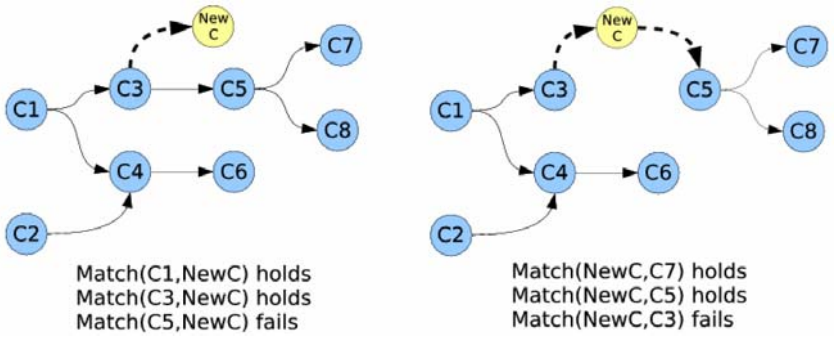


Figure II-41 : Exemple de construction d'un graphe des services.

Une fois le graphe construit et disponible, l'algorithme de recherche sélectionne les îlots utilisant les ontologies requises puis teste les services en tenant compte des relations de correspondance. Si un service (nœud) ne correspond pas à celui demandé, inutile de chercher les services correspondants (les descendants du nœud) (Figure II-42).

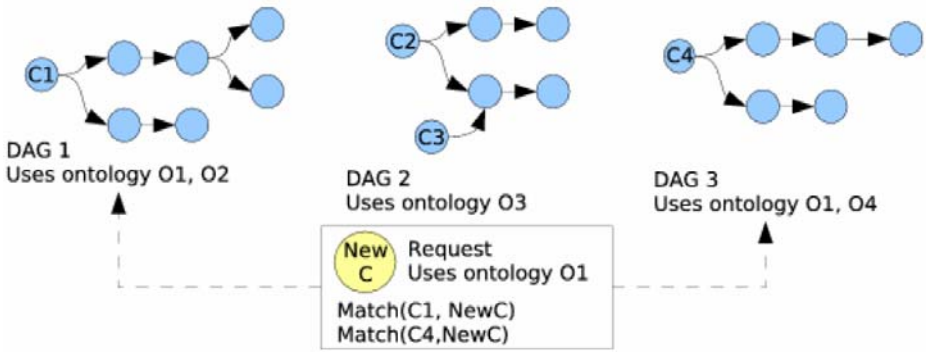


Figure II-42 : La sélection des îlots à explorer se fait en fonction des ontologies utilisées.

Conclusion sur SD-SDCAE

SD-SDCAE permet de prendre en compte la sémantique d'un service. La recherche s'appuie sur une fonction de matchmaking sensiblement différente de [83 Paolucci 2002]. Elle se base sur la distance sémantique entre services. Les services sont généraux. Ils n'intègrent absolument pas l'IHM. Outre ce fait, examinons les limites de l'approche.

Le site http://www.daml.ri.cmu.edu/matchmaker/ permet de tester une approche similaire à SD-SDCAE pour la définition et la recherche de services. Il donne une illustration de service sur un calcul d'itinéraire entre deux aéroports à une date donnée. Le modèle du service est partiellement reproduit ci après (Tableau II-21) J'utilise cet exemple pour illustrer les limites de l'approche.

Tableau II-21 : Description d'un service de réservation de vol entre des aéroports de départ et d'arrivée.

Ontology	http://www.daml.ri.cmu.edu/matchmaker/owl/Concepts.owl	
Category	**Airline** (UNSPSC_10223525 et NAISC_525)	
Inputs	DepartureAirport	**Airport**
	ArrivalAirport	**Airport**
	Departure	**Date**
Outputs	Flight Itinerary	**FlightItinerary**

L'algorithme de correspondance (matchmaking) de SD-SDCAE ne permet de discriminer les entrées que par leurs types. Ainsi, deux paramètres de même type, mais ayant des rôles différents, ne sont pas distinguables (idem pour deux résultats). Sur l'exemple, c'est le cas des aéroports de départ et d'arrivée. J'examine ci-dessous des solutions et cerne leurs limites.

- Affinage des concepts utilisés dans les interfaces : Cela consisterait à définir des sous concepts « Aéroport de départ » et « aéroport d'arrivée ». Fondamentalement, cette solution mélange la « nature » d'un concept (la notion d'aéroport) et son rôle (départ / arrivée). J'y vois deux inconvénients majeurs :
  - Premièrement, on peut s'attendre à une explosion du nombre de concepts, par la diversité des rôles possibles (« où on passe », « où on ne passe pas »…). Chaque nouveau rôle impliquerait un changement de l'ontologie, ce qui en plus poserait des problèmes de maintenance.

- Deuxièmement, et plus fondamentalement, tout service prenant en entrée deux aéroports (sans préciser leurs rôles) sera reconnu et il ne sera pas possible de déterminer lequel des deux est celui de départ…

- Utilisation des noms d'instances : Cela consisterait à différencier les concepts d'un même type par leur nom de variable (ex : aéroport-d-arrivée versus aéroport-de-départ). Cette solution semble pouvoir résoudre le problème mais elle a deux inconvénients :
  - Tout couple <nom, concept> (ex : <aéroport-d-arrivée, Aéroport>) doit conduire à une interprétation unique. Rien ne le garantit formellement.
  - Une idée (la notion d'aéroport) doit conduire à un concept unique.

On se heurte en fait ici aux limites du raisonnement sur les ontologies. On peut modéliser une certaine sémantique mais pas toute la sémantique nécessaire. Comme on le voit, ces limites ne sont pas simples à dépasser. Une piste à creuser pourrait être les graphes conceptuels [93 Sowa 1984].

Modélisation de la sémantique à l'aide des graphes conceptuels

Les graphes conceptuels sont un type de réseau sémantique. Un réseau sémantique est un système de représentation graphique des connaissances basé sur des noeuds interconnectés par des arcs. Il prend donc la forme d'un graphe. Un graphe conceptuel est un graphe biparti étiqueté par des items lexicaux. Une des classes de sommets correspond à des concepts ; l'autre à des relations entre concepts. C'est l'une des forces des graphes conceptuels : ils permettent de représenter non seulement les concepts mais aussi les liens entre concepts.

On peut associer à un graphe conceptuel une formule de la logique du premier ordre. Il existe une sémantique logique consistante et complète, équivalente à la logique du premier ordre, pour le modèle des graphes conceptuels comme l'ont montré [93 Sowa 1984] et [23 Chein 1992].

Le terme de graphe conceptuel proposé par Sowa définissait à la fois un modèle de base assez précis (appelé graphe conceptuel simple) et des extensions (ou idées d'extensions) de manière plus ou moins formelle. Des chercheurs ont depuis développé ce modèle de base, comme [24 Chein 1996] qui ont notamment proposé la notion de graphes emboîtés. Un graphe emboîté est tel que les noeuds concepts peuvent inclure d'autres graphes. Comme le précisent [24 Chein 1996], les graphes conceptuels peuvent être considérés comme :

- un modèle déclaratif de représentation des connaissances.
- un moyen de résoudre les calculs d'inférence par des algorithmes de graphe.

L'avantage des graphes conceptuels sur la logique du premier ordre ne se situe pas au niveau de leur puissance d'expression brute (elle est équivalente) mais principalement au niveau de leur lisibilité.

Les connaissances exprimées dans un graphe conceptuel n'ont de sens que par rapport à un support donné. Un support définit le vocabulaire de base. Il comprend un ensemble ordonné (treillis) de concepts et de relations (un treillis pour les concepts et un treillis par arité de relations), chaque relation étant munie d'une signature. Les ordres sur ces ensembles sont des liens de spécialisation.

L'opération de projection : l'équivalent du matchmaking

Grâce à l’opération de projection offerte par les graphes conceptuels, on peut savoir si l’information présente dans un graphe est déductible d’un autre. La projection d’un graphe G dans un graphe H revient à chercher si l’information portée par G peut être déduite de celle portée par H.

Limites des graphes conceptuels

Les graphes conceptuels sont plus puissants que les simples ontologies mais souffrent malgré tout de limites. La plus fondamentale est qu'à un graphe donné correspondent en général de nombreux graphes synonymes. Rien ne garantit l'unicité de représentation d'une idée en graphe conceptuel. Pire encore, les graphes synonymes ne se projettent pas forcément les uns dans les autres. La Figure II-43 montre deux graphes pouvant être interprétés comme sémantiquement équivalents.

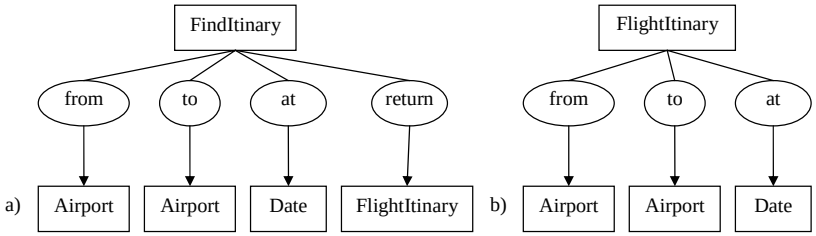


Figure II-43 : Différentes modélisations (a et b) du service de réservation d'avion à base de graphes conceptuels.

Les graphes conceptuels n'apportent donc pas de solution « miracle » au problème de la modélisation de la sémantique.

IV. Conclusion de l'état de l'art

Cet état de l'art a montré comment se situait l'approche BàO d'interacteurs plastiques par rapport aux autres approches visant à résoudre la plasticité des IHM : IDM et OS. L'approche BàO n'est pas redondante. Au contraire, elle est complémentaire. En effet, en fournissant des interacteurs plastiques, elle permettrait à des travaux comme Façade (approche OS) de réaliser la substitution de présentation bien plus facilement et élégamment. De même, les approches basées modèles gagneraient à utiliser des interacteurs plastiques d'un point de vue méthodologique et de l'exécution.

Aucune des BàO d'interacteurs plastiques ne réunit tous les requis. En particulier, aucune d'entre elles ne considère les opérateurs entre tâches (ou les décorations de tâches) comme des interacteurs. Une telle approche permettrait de modéliser toute la sémantique du modèle de tâches à l'aide d'interacteurs plastiques. Malgré cela, on retrouve des caractéristiques intéressantes : la notion de shell dans FRUIT, de contexte généralisé dans UBIT, d'output modules dans MultiModal Widgets, de Presenters dans ACE.

L'état de l'art a également permis d'avoir une compréhension des domaines connexes aux BàO : les langages de transformation et les annuaires de services. Ces outils sont utiles pour transformer, capitaliser et retrouver dynamiquement des IHM. L'état de l'art sur ces deux domaines n'est sans doute pas complet mais il en ressort que : il n'existe pas de langage de sélecteur simple et puissant à la fois ; il n'existe pas de façon univoque d'exprimer la sémantique d'un service et donc le classement et la découverte automatique de services, même si des pistes existent.

Mes contributions s'appuient sur les enseignements de cet état de l'art.

Contributions logicielles

Ce chapitre décrit mes contributions logicielles. Elles s'articulent autour de la notion de COMET (**CO**ntext **M**ouldable widg**ET**), un interacteur façonné pour la plasticité. Une première section pose les bases de ma vision : le système interactif dans son contexte d'usage est décrit par un graphe de modèles. Ce graphe décrit l'écosystème, c'est-à-dire le système interactif, son contexte d'usage et le déploiement du système interactif dans son contexte d'usage. Par abus de langage, on appellera Ecosystème ce graphe de modèles. Il fait l'objet de la section 1. Je propose ensuite, en section 2, un modèle d'architecture logicielle et une boîte à outils pour la plasticité : les COMET. Ces propositions sont dirigées par mes objectifs et la revue critique de l'état de l'art. En particulier, une COMET est définie au niveau tâche utilisateur ; elle est extensible et multi-technologies. Son extensibilité s'appuie sur un annuaire de systèmes interactifs appelé GDD (section 3) dans lequel la COMET puise des présentations sur mesure. Un système interactif est un graphe de COMET sur lequel des transformations sont applicables. Je propose un langage de style, CSS++, pour la manipulation du graphe de COMET (section 4).

La Figure III-1 situe l'ensemble de mes contributions logicielles sur ma vision de la plasticité.

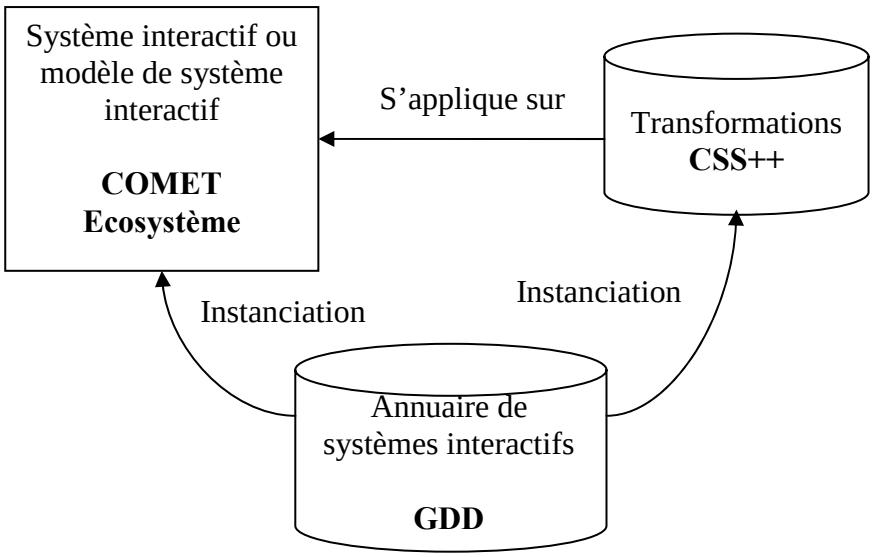


Figure III-1 : Cartographie de mes contributions logicielles.

La section suivante est consacrée à l'Ecosystème.

I. Ecosystème : Graphe de modèles pour la plasticité

Cette contribution a donné lieu aux publications suivantes : [39-40 Demeure 2005], [9 Balme 2006], [19 Calvary 2006], [91 Sottet 2006], [29 Coutaz 2007], [92 Sottet 2007].

Les IHMs plastiques doivent faire face à la variété, variabilité et imprévisibilité du contexte d'usage. Pour cela, je propose de considérer le système interactif « en contexte », c'est-à-dire d'assembler en un même graphe (l'écosystème) la description du système interactif, du contexte d'usage et du déploiement de l'un sur l'autre. Chaque modèle représente un point de vue particulier sur le système interactif. L'utilisation de différents modèles pour raisonner sur les IHM est une idée largement répandue [102 Vanderdonckt 1999] [72 Limburg 2004] [90 Sottet 2005] [25 Clerckx 2004]. Mais pour être productif (interprété et manipulé par une machine), chaque modèle doit être accompagné d'une description précise et explicite : c'est la notion de métamodèle en IDM. Le développement d'interfaces classiques implique la mise en place de cinq métamodèles : Concepts, Tâches, AUI, CUI et FUI [90 Sottet 2005]. Pour la plasticité, trois métamodèles supplémentaires sont nécessaires : plate-forme, environnement et utilisateur. Les liens entre modèles modéliseront, en particulier, le déploiement du système interactif dans son contexte d'usage. J'examine ces modèles ci-dessous :

- C&T : Décrit les concepts du domaine, les tâches utilisateur et les liens entre concepts et tâches. Le modèle de C&T exprime la sémantique du système du point de vue de l'utilisateur. Les métamodèles qui peuvent être utilisés pour décrire le niveau C&T sont, entre autres, UML, [84 Paterno 1997], MAD [89 Scapin 1990], etc.
- AUI : Modélise le dialogue en terme d'espaces de dialogue, de navigation entre espaces et de hiérarchie d'espaces (ex : l'espace E1 est inclus dans l'espace E2). Le modèle d'AUI est plus précis que le modèle des tâches. Par exemple, un entrelacement peut se traduire de différentes façons en terme d'AUI selon le mode d'accès aux espaces entrelacés. Les métamodèles d'AUI peuvent être, entre autres, les automates, les statecharts, les réseaux de pétri, etc.
- CUI : Les modèles de CUI introduisent des considérations géométriques, spatiales, temporelles, etc. (selon la modalité utilisée). De nombreux langages existent qui permettent de représenter les CUI : XHTML, SWING, TK, graphes de scènes, etc.
- Noyau fonctionnel (FC) : Les modèles de FC représentent le système d'un point de vue programmatique. Le but est de décrire l'interface programmatique des systèmes, de modéliser les états du système, etc. Des langages comme WSDL ou OWL-S permettent cela.
- Plate-forme : Le modèle de plate-forme représente les différents aspects d'une plate-forme informatique : CPU, mémoires, réseau, dispositifs d'entrée/sortie, programmes en cours d'exécution, etc. Des langages de modélisation de plate-forme existent comme dans CC/PP ou Amigo.

- Environnement : L'environnement est une description de l'environnement physique (bruit, luminosité, objets physiques, animaux, humains, ordinateur, etc.). Un moyen de représenter l'environnement est, par exemple, d'utiliser des modèles 3D.
- Les utilisateurs : Ce modèle exprime les caractéristiques des utilisateurs utiles pour le système : niveau d'expertise pour une tâche donnée, âge, handicap, etc.
- Relations : Tous les modèles précédemment décrits donnent un point de vue particulier sur le système interactif. Le but du modèle de relations est d'établir les relations existant entre ces descriptions.

Tâches et concepts sont reliés au modèle de noyau fonctionnel. L'interface concrète est quant à elle reliée au modèle de la plate-forme via trois types de relations : elle est exécutée sur une plate-forme, rendue sur des dispositifs de sortie et accessible via des dispositifs d'entrée. Les utilisateurs sont en lien avec ces mêmes dispositifs. Enfin, modèles d'utilisateur et de plate-forme sont liés au modèle d'environnement (Figure III-2).

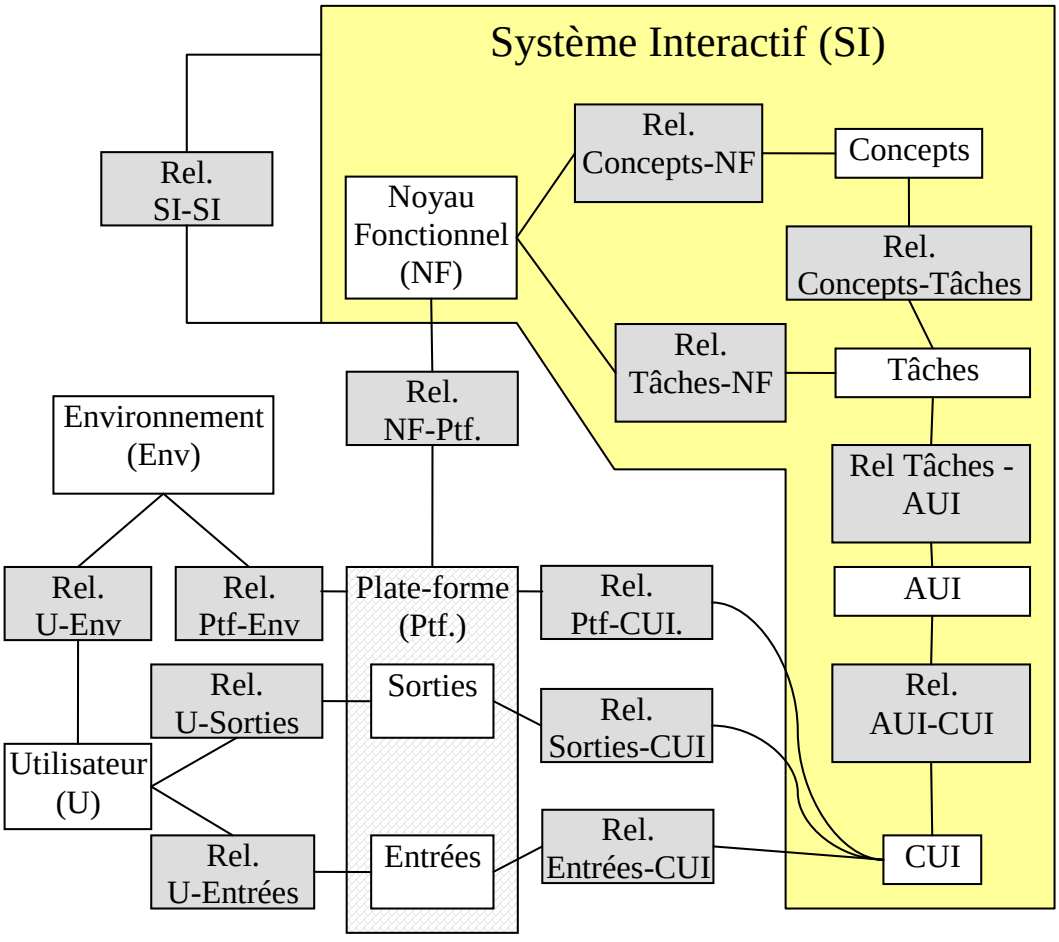


Figure III-2 : Ecosystème : les boîtes blanches représentent des (méta)modèles ; les boites grises des relations (mappings) entre (méta)modèles.

L'Ecosystème permet de cerner l'impact d'une modification dans un modèle. Par exemple, la modification d'une tâche impliquera des changements au niveau de l'ensemble des interacteurs auxquels elle est liée. Autre exemple, le déplacement des utilisateurs ou des dispositifs de rendu dans l'environnement pourrait rendre

inaccessible une tâche particulière (par exemple, la consultation d'une information) si les interacteurs correspondants n'étaient pas grossis ou remplacés par de plus adaptés.

Reprenons l'exemple de Sedan-Bouillon pour illustrer l'Ecosystème (cf l'introduction). La Figure III-3 donne une représentation simplifiée, pour des raisons de lisibilité, de l' Ecosystème de Sedan-Bouillon à un instant *t*. Cet instant *t* correspond dans le scénario au moment où Lionel suggère à Alex d'utiliser son téléphone portable. En haut de la Figure III-3, se trouve le modèle des tâches CTT de Sedan-Bouillon. Au milieu de la figure, se trouvent les modèles de plates-formes (PC, PDA, TEL pour téléphone). Sur la figure, la représentation du PC et du PDA contiennent des représentations de la CUI et de l'AUI de Sedan-Bouillon pour exprimer le mapping de ces éléments sur ces plates-formes. Les relations entre l'AUI et la CUI sont implicites. Les plates-formes sont reliées à des dispositifs d'entrée et de sortie : ce sont les dispositifs qu'elles gèrent. Les utilisateurs Lionel et Alex sont en contact avec certains de ces dispositifs. Cette notion de contact est représentée par la proximité géographique entre les représentations des utilisateurs et des dispositifs. A un modèle d'utilisateur (ex : Lionel) ou un modèle de dispositif (ex : écran) correspond une zone de l'espace dans le modèle de l'environnement (zone basse de la figure). Cela représente à la fois le modèle d'utilisateur/plate-forme et leur position spatiale dans l'environnement. Enfin, des mappings sont explicitement représentés en gras entre le modèle de tâches et le modèle d'AUI.

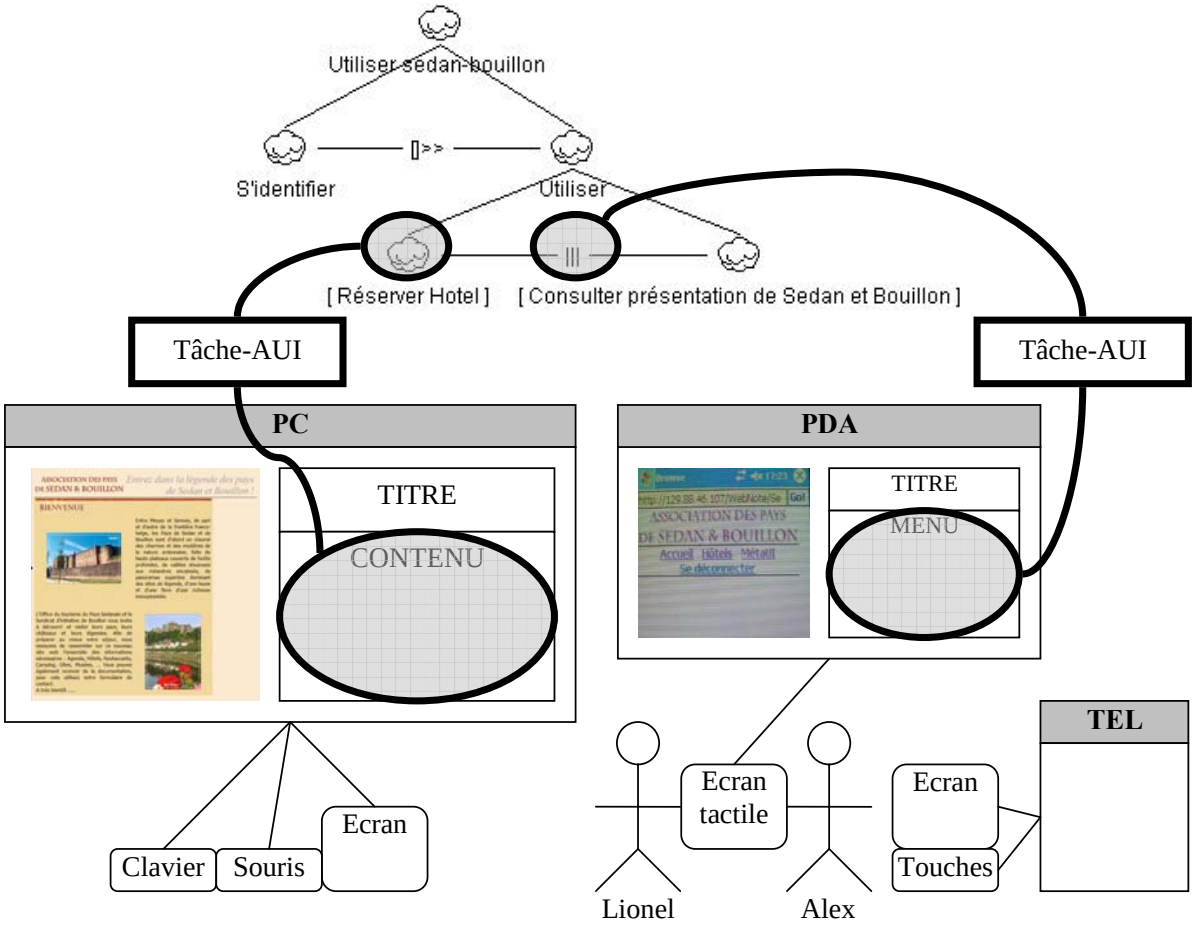


Figure III-3 : Ecosystème de Sedan-Bouillon à un instant *t*.

L'Ecosystème ne présage pas de la façon dont le système interactif sera implémenté. Il peut être développé par des techniques IDM mais pas nécessairement. Il peut être

implémenté par des interacteurs plastiques (par exemple, les COMET) mais pas nécessairement.

Voyons maintenant comment cet Ecosystème peut être utilisé pour raisonner sur l'état du système interactif. Dans le scénario, lorsqu'Alex éteint le PDA, l'opérateur entre les tâches « Réserver Hotel » et « Consulter présentation de Sedan-Bouillon » n'est alors plus rendu. En conséquence, la navigation entre ces deux sous tâches ne se fait plus. Un tel état est immédiatement visible sur l'Ecosystème.

De façon générale, détecter de tels cas, c'est repérer des états particuliers de l'écosystème : ici, l'état où une partie des tâches (ou opérateurs, décorations, etc.) n'est gérée par aucune plate-forme. La Figure III-4 propose un algorithme permettant de détecter un tel état. L'algorithme est en pseudo Z. Il détecte aussi les tâches qui, même gérées par une plate-forme, ne seraient pas rendues sur un dispositif de sortie. Le principe est de suivre les relations existantes entre le modèle de tâches et les dispositifs d'affichage (tâche→AUI→CUI→plate-forme→dispositifs de rendu).

bool ECOSYSTEME::ToutesTâchesGérées? ( )
→ SontAffichées?({t : ECOSYSTEME::Tasks | t.active • t})
// ECOSYSTEME::Tasks désigne toutes les tâches. On sélectionne seulement les tâches actives (celles en cours)

bool ECOSYSTEME::SontAffichées? (ST:set of Tasks)
→ ∀t : ST • ∃m : ECOSYSTEME::RelTâcheAUI | t ∈ m.Tasks ∧ SontAffichées? (m.AUI)
// Un ensemble de tâches est affiché si chaque tâche est reliée à un AUI qui est lui-même affiché.

bool ECOSYSTEME::SontAffichées? (S_AUI: set of AUI)
→ ∀a : S_AUI • ∃m : ECOSYSTEME::RelAUI-CUI | a ∈ m.AUIs ∧ SontAffichées? (m.CUI)
// Un ensemble d'AUI est affiché si chaque AUI est reliée à une CUI qui est elle-même affichée.

bool ECOSYSTEME:: SontAffichées? (SUI: set of CUI)
→ ∀ ui : SUI • cpu = CPU(ui) ∧ (∃ d : cpu.Displays | d. Affiche?(ui))
// Un ensemble de CUI est affiché si chaque CUI est exécutée sur une plate-forme (CPU) et si cette plate-forme rend effectivement la CUI sur un de ses dispositifs de rendu..

Figure III-4 : Exemple de méthodes pour raisonner sur un Ecosystème.

Les méthodes énoncées ci-dessus permettent de raisonner sur l'état du système interactif à un instant donné. Parfois, il est nécessaire de raisonner sur un ensemble d'états. C'est typiquement le cas de la migration pour laquelle il convient de comparer deux états du système : avant et après migration. La Figure III-5 montre comment, pour un ensemble de tâches données, identifier les plates-formes qui ne gèrent plus de tâches et celles qui commencent à en gérer.

// Migration retourne un couple d'ensemble de plates-formes : les plates-formes quittées et les plates-formes rejointes.
<set of Platforms, set of Platforms>Migration (ST : set of Tasks; *// Les tâches considérées*
E1, E2 : ECOSYSTEME) *// Les deux états considérés*
SP1 = E1. PtfGerantes (ST) // Les plates-formes qui géraient les tâches
SP2 = E2. PtfGerantes (ST) // Les plates-formes qui gèrent les tâches
→ <SP1 \ SP2, SP2 \ SP1> *// < plates-formes qui ne prennent plus de tâches en charge*
, plates-formes qui ne prenaient pas de tâches en charge et qui en prennent maintenant>

Figure III-5 : Exemple de fonction pour raisonner sur deux Ecosystèmes.

L'Ecosystème permet de raisonner sur l'éparpillement d'un système interactif. Je propose ici quelques mesures de cet éparpillement :

- Le nombre de dispositifs affichant tout ou partie du système interactif. Cela donne une indication quant à la charge de travail de l'utilisateur pour

percevoir l'état du système. On peut en effet supposer que, plus il aura de dispositifs de rendu à surveiller, plus la charge sera grande.

- L'ensemble E des ensembles de dispositifs affichant l'intégralité de l'état courant du système. Par exemple, on peut imaginer l'affichage redondant (vue multiple) d'un site sur un PC et un PDA (premier ensemble E1), un autre PC et un téléphone (deuxième ensemble E2) ou un PC (troisième ensemble E3). Le cardinal de E et l'intersection des ensembles Ei donnent des indications quant à la redondance de l'éparpillement.
- L'ensemble SC des couples <tâche, ensembles de dispositifs représentant/manipulant cette tâche>. Cela donne une indication quant à l'utilisation des dispositifs :
  - L'assignation [27 Coutaz 1995]: un dispositif est dédié à une tâche donnée. Ceci se formalise en : $\forall c1,c2: SC \mid c1 \neq c2 \bullet c1.disps \cap c2.disps = \varnothing$
  - Absence de redondance totale : il n'existe pas deux dispositifs gérant le même ensemble de tâches. Ceci se formalise en : $\forall c1,c2: SC \mid c1 \neq c2 \bullet c1.disps \neq c2.disps$
- La disposition spatiale des dispositifs, notamment de sortie, et leurs relations avec les CUI et les tâches. Ceci permet en particulier de détecter les « mosaïques d'écrans » [97 Tandler 2001] [56 Lachenal 2004].

En synthèse, l'Ecosystème permet de raisonner sur le système interactif dans son contexte d'usage. D'autres propriétés ou opérations pourraient être définies, prenant en compte des préoccupations particulières. Par exemple, en utilisant l' Ecosystème, il est possible de savoir si deux interacteurs correspondent à une même tâche ou non.

La Méta-IHM illustrée dans Sedan-Bouillon est une IHM de l'Ecosystème. En particulier, les cases à cocher permettent le contrôle des mappings entre les espaces de dialogue et les plates-formes. La Méta-IHM est à l'étude. [28 Coutaz 2006] et [88 Roudaut 2006] distingue les Méta-IHM tissées ou non tissées. Une Méta-IHM tissée est mêlée à l'IHM. Le Tableau III-1 donne quelques exemples. Il distingue les Méta-IHM tissées des Méta-IHM non tissées. Il distingue aussi les Méta-IHM qui agissent directement sur l'IHM versus sur une représentation de l'IHM.

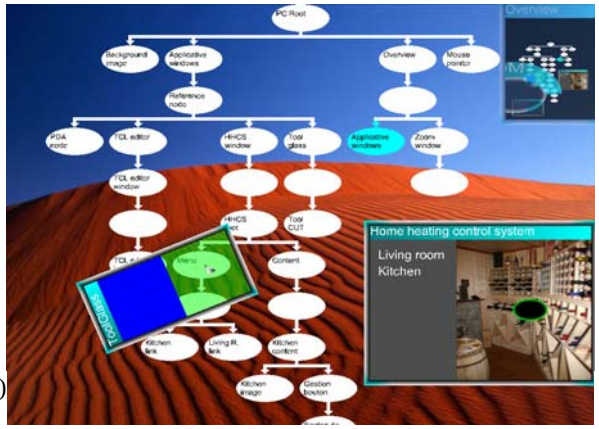
Tableau III-1 : Exemples de Méta-IHM.

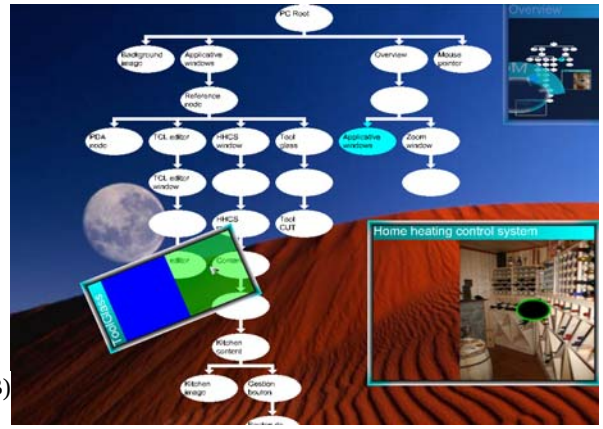
Méta-IHM	Tissées	Non tissées
Agit sur l'IHM	Barre de manipulation des fenêtres. Barres d'outils attachables. Collapse zoom [11 Baudisch 2004].	Barre des tâches Windows. ToolGlass sur la scène.
Agit sur un modèle de l'IHM	Sedan-Bouillon.	XXL [64 Lecolinet 1996], ToolGlass sur mode.

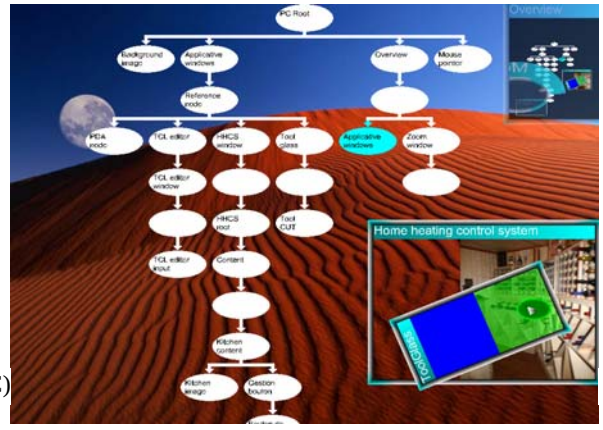
La Figure III-6 montre la Méta-IHM de Sedan-Bouillon affichée sur un PC. La méta-IHM est tissée à l'IHM : c'est un des menus de la barre de navigation. Elle manipule une représentation de l'Ecosystème.

La Figure III-7 montre une Méta-IHM à base de Toolglass. Elle est non tissée à l'application. L'utilisateur peut agir directement sur l'IHM (image de fond, fenêtre, etc.) ou sur une représentation de celle-ci (ici, le graphe de scène de l'application).

Figure III-6 : Exemple de Méta-IHM tissée (Sedan-Bouillon).

A) 

B) 

C) 

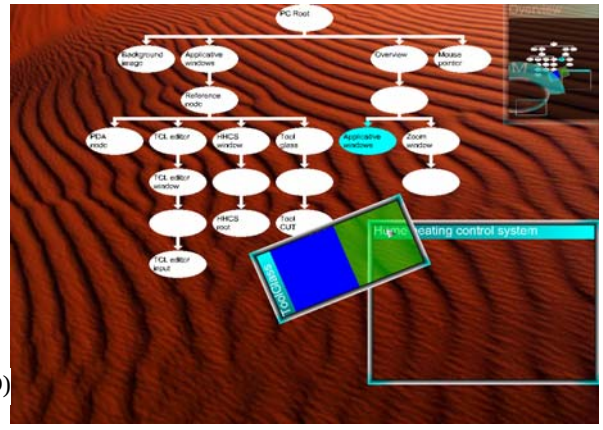
D) 

Figure III-7 : Exemple de Méta-IHM non tissée.

I.1. Conclusion sur l'Ecosystème

L'Ecosystème est un moyen puissant de décrire et de raisonner sur les systèmes interactifs déployés dans leur contexte d'usage. Il est possible d'exprimer des propriétés et de réaliser des opérations sur l'Ecosystème. La Méta-IHM est une représentation de l'Ecosystème. Elle peut être tissée à l'IHM ou non. Elle peut agir directement sur l'IHM ou sur une de ses représentations.

L'Ecosystème ne présage pas de la façon dont sera construit le système interactif. Les approches IDM peuvent être une solution. Elles modélisent naturellement le système interactif selon différentes perspectives. J'adopte une approche complémentaire : les BàO d'interacteurs plastiques. Ces interacteurs, les COMET, se décrivent (aujourd'hui partiellement) selon tous les points de vue de l'Ecosystème. COMET s'appuie sur un style d'architecture logicielle qui fait l'objet de la section suivante.

II. COMET : Interacteurs plastiques

Cette contribution a donné lieu aux publications suivantes : [33 Daassi 2003], [17-18 Calvary 2004], [37 Demeure 2004], [20 Calvary 2005], [41-42 Demeure 2006].

COMET est un style d'architecture logicielle. Il donne lieu à une BàO d'interacteurs plastiques du même nom : les COMET. J'en donne ici une description conceptuelle puis implémentationnelle.

II.1. Description conceptuelle

II.1.A. Style d'architecture

Un style d'architecture [26 Coutaz 1993] requiert la définition d'un vocabulaire. Ce vocabulaire fixe les éléments de conception. Par exemple, PAC est un style d'architecture qui s'appuie sur trois éléments clé : les facettes abstraction (A), présentation (P) et contrôle (C). C'est le vocabulaire du style PAC. Un style nécessite la spécification des contraintes de configuration entre ces éléments. Par exemple, dans PAC, les facettes A et P ne communiquent pas directement : les échanges transitent par C. C communique directement avec A, P et les facettes C de ses père et fils. Ce sont les contraintes de configuration du style PAC. Le style fixe la sémantique du modèle. Par exemple, dans PAC, A représente l'abstraction. Elle est indépendante de tout choix de représentation. COMET est un style d'architecture logicielle pour la plasticité. J'en précise ici le vocabulaire, la sémantique et les contraintes de configuration. J'illustre sur un exemple simple : la tâche utilisateur « Se connecter », dite « Log ». L'identification de l'utilisateur peut s'effectuer de différentes manières : par un couple <identifiant, mot de passe>, par la reconnaissance de la voix ou du visage, etc.

Vocabulaire et sémantique

Le style COMET est dirigé par deux grands principes. Premièrement, une COMET est un système interactif. On doit donc pouvoir la considérer selon plusieurs points de vue, en particulier le point de vue de la présentation et le point de vue fonctionnel. Deuxièmement, les COMET ne visent pas à remplacer les BàO d'interacteurs existantes. Elles doivent, au contraire, en tirer profit, c'est-à-dire s'appuyer sur ces BàO pour les mises en œuvre technologiques.

La Figure III-8 donne en a) une modélisation UML d'une COMET. L'équivalent dans notre représentation graphique est donné en b). Les éléments sont détaillés à la suite de la figure.

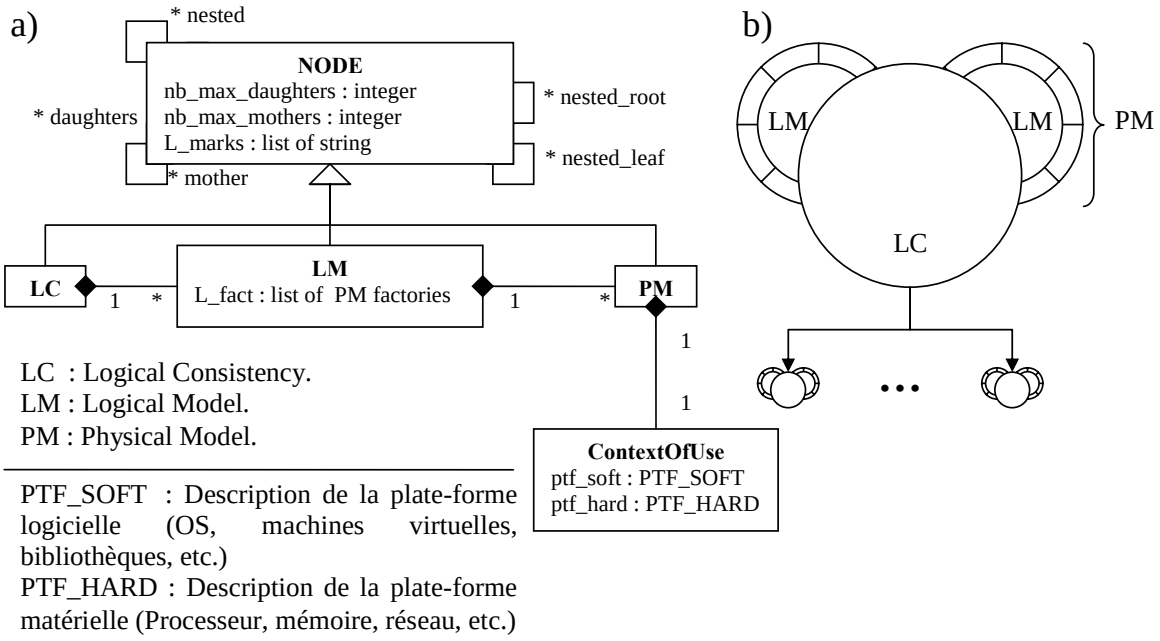


Figure III-8 : a) Modèle UML de COMET. b) Représentation graphique.

Une COMET est constituée de trois types d'éléments. Chaque type est en charge d'une préoccupation particulière :

- Un LC (pour Logical Consistency) représente la tâche utilisateur (spécifier, s'identifier, etc.) ou l'opérateur entre tâches (entrelacer, séquencer, etc.) offert par la COMET. C'est l'élément central de la COMET : sa « raison d'être ». En d'autres termes, le LC porte la sémantique du « service » rendu par la COMET. Cette sémantique se traduit, en particulier, par une API dite « sémantique ». Cette sémantique se décline en différents modèles logiques (LM). Le LC est chargé de maintenir la cohérence logique (d'où son nom) entre les différents LM.
- Des LM (pour Logical Model) sont associés à ce LC. Un LM est chargé d'une préoccupation particulière dans la réalisation du LC. Les exemples classiques de LM sont les LM de présentation chargés de présenter la tâche à l'utilisateur (cf la facette P de PAC) et les LM fonctionnels chargés de la mise en œuvre algorithmique du LC indépendamment de toute présentation (cf la facette A dans PAC). Chaque LM doit implémenter l'API sémantique du LC : c'est le langage qu'ils partagent. Le LM peut éventuellement étendre cette interface pour prendre en compte des considérations qui lui sont propres. Le LM est chargé de maintenir une cohérence entre les différents modèles physiques (PM) qui lui sont liés. Il fournit de plus des usines à PM pour l'instanciation de nouveaux PM.
- Des PM (pour Physical Model) sont associés à ces LM. Un PM est associé à un seul LM. Un PM est une façon particulière de réaliser un LM. Un PM encapsule du code de BàO. Par exemple, un PM de présentation associé à la COMET de Log peut encapsuler le code correspondant à des champs textes de la BàO HTML ou TK, SWING, etc. Autre exemple, un PM fonctionnel associé à une COMET de messagerie instantanée peut encapsuler le code correspondant au protocole IRC ou MSN, AIM, etc. Ce code de la BàO utilisé dans le PM est appelé « primitives technologiques ». Le PM doit

implémenter l'API sémantique du LM qui lui est associé : c'est leur « langage » commun. Il fournit de plus une description du contexte d'usage qui lui est nécessaire. Par exemple, il exprime le fait qu'il lui faille une plate-forme matérielle de type PC et logicielle de type Java 1.4. Une catégorie de PM particulière est celle des PM dits « universels ». Un PM universel n'a aucune exigence en terme de contexte d'usage : il peut être branché n'importe où. Une fois branché, le PM universel prend comme contexte d'usage celui du PM père. Dès qu'il est débranché, le PM universel reprend son contexte d'usage vierge, ie sans contrainte. Les connections entre éléments font l'objet de la section suivante.

J'appelle **Nœud** tout LC, LM ou PM. Un nœud peut être marqué. Par exemple, un LC peut être marqué de l'estampille Fréquence ou Criticité pour embarquer les décorations classiquement posées sur les tâches. Un LM peut être marqué de la préoccupation qu'il gère (par exemple, présentation). Un PM peut être marqué de la longueur de la trajectoire d'interaction nécessaire à chaque action utilisateur.

Contraintes de configuration

Une COMET est définie par un LC. Un LC est connecté à un ou plusieurs LM. Le LC et les LM qui lui sont connectés communiquent bidirectionnellement. Un LM est connecté à un ou plusieurs PM. Le LM et les PM qui lui sont connectés communiquent, de même, bidirectionnellement. Par exemple, dans le cas du Log, si le concepteur a jugé pertinent de distinguer abstraction et présentation, alors deux LM sont créés : un pour l'abstraction ; l'autre pour la présentation. Plusieurs présentations sont envisageables (Figure III-9) selon que l'utilisateur se connecte par identifiant/mot de passe ou se laisse reconnaître par sa voix ou son visage. Elles sont implémentables dans différentes technologies. Chaque implémentation donne lieu à un PM.

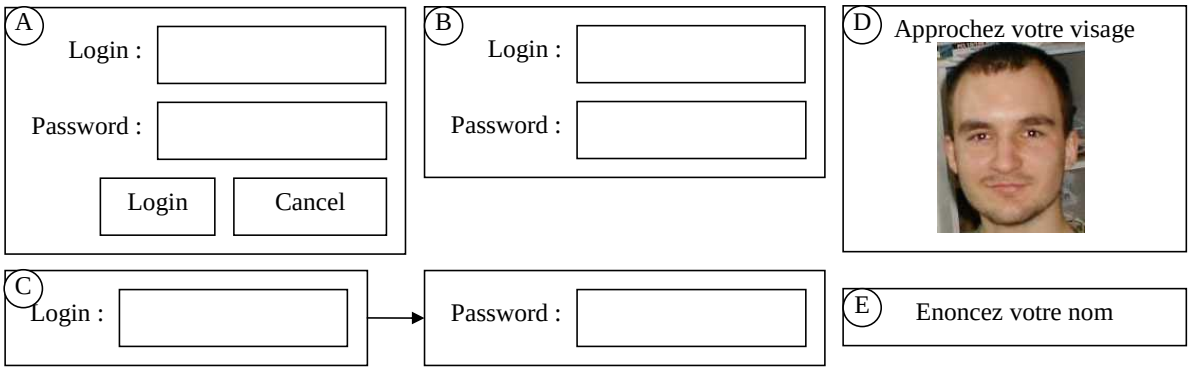


Figure III-9 : Des représentation possibles de la COMET Log.

D'un point de vue de l'abstraction, il existe différentes manières de vérifier l'identité de l'utilisateur : vérification du couple <identifiant, mot de passe>, analyse d'image, reconnaissance de voix, etc. Le choix dépend, en partie, du PM utilisé. Mais on pourrait aussi imaginer différents algorithmes de reconnaissance de visages par exemple, présentant des propriétés différentes en temps de calcul, précision, etc. C'est le LC qui gère la compatibilité entre les LM d'abstraction et de présentation. Si, par exemple il n'y a pas de PM fonctionnel permettant une analyse de visage, alors le LC proscrira au LM de présentation l'utilisation d'un tel PM.

La Figure III-10 propose une représentation graphique pour le style COMET. Elle est appliquée au Log.

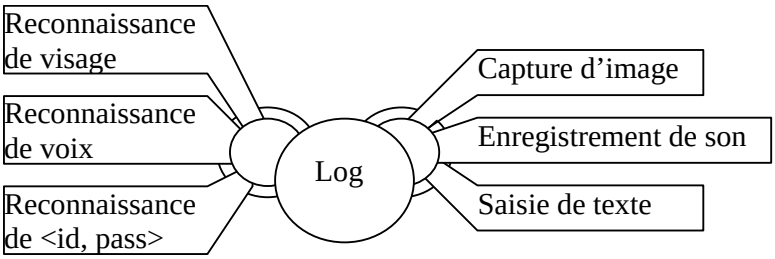


Figure III-10: Représentation graphique de la COMET Log.

En plus de la compatibilité entre les différents types de PM, le LC gère la cohérence de la COMET. Supposons que l'API sémantique du Log soit la suivante (Figure III-11) :

- reset() : Réinitialise les informations de log.
- add_info_log(string type, void*) : Ajoute une information de log. Cette information est typée (type) et désignée par un pointeur (void*). Des exemples d'informations de log sont : un couple <identifiant, mot de passe>, une image ou encore un enregistrement sonore. Le type permet d'identifier des PM fonctionnels compatibles des PM de présentation.
- get_info_log() : Renvoie la liste des informations de log.
- log() : Tente de se connecter compte tenu des informations de log fournies. Après l'appel à la méthode Log du LC, les informations de log sont réinitialisées.

API sémantique du Log
reset() : void **add_info_log(string, void*) : void** **get_info_log() : list of <string, void*>** **log() : bool**

Figure III-11 : API sémantique de la COMET de Log.

Prenons le cas de deux méthodes :

- reset : Cette méthode peut être déclenchée directement au niveau du LC (Figure III-12-A). Le LC propage l'appel à ses LM qui le propagent à leurs PM. Notons que selon la méthode considérée, l'ordre d'appel aux LM peut avoir de l'importance.
- log : Cette méthode peut être déclenchée à partir d'un PM (Figure III-12-B). Par exemple, un PM de saisie d'un couple <identifiant, mot de passe>. Le PM propage l'appel à son LM. Celui-ci détermine dans quelle mesure l'appel doit être propagé aux autres PM. Il peut y avoir différentes politiques à ce niveau. L'utilisateur doit-il se logger sur chaque PM ou bien un seul suffit-il ? Une fois cela déterminé, le LM transmet l'appel à son LC qui le transmet aux autres LM. Ces LM le transmettent à leur tour à leurs PM.

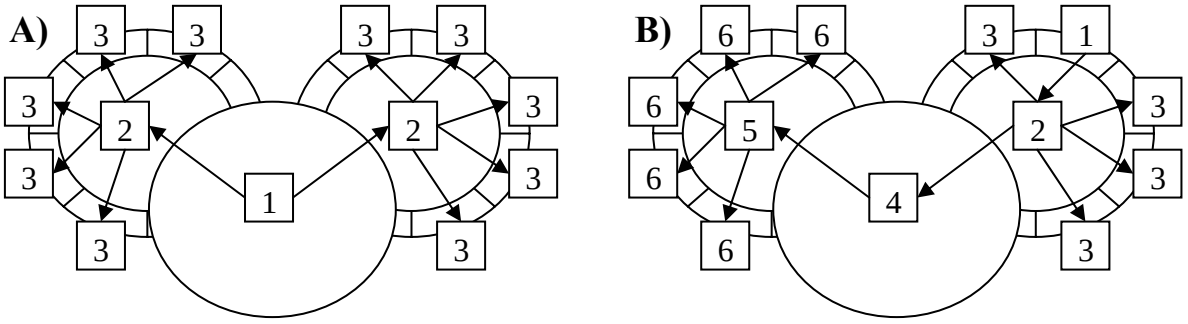


Figure III-12 : Propagation des appels de méthode pour la cohérence de la COMET. Les numéros représentent l'ordre d'appel. Les flèches indiquent le sens de propagation.

Les COMET s'assemblent sous forme de graphes orientés. Plus précisément, elles donnent lieu à trois graphes parallèles : le graphe des LC, le graphe des LM de présentation et le graphe des PM de présentation. Ces graphes sont orientés : la relation « père-fils » signifie que le fils s'exprime dans le père. Par exemple, dans le cas de PM de présentation graphique, « PMF est fils de PMP » signifie que l'affichage de PMF dépend de PMP.

Reprenons la COMET Log mais cette fois enrichie d'une image et d'un texte pour indiquer à l'utilisateur l'application à laquelle il est en train d'essayer de se connecter. La (Figure III-13) b) prend l'exemple de clubic.com. Le graphe sémantique des COMET (graphe des LC) est donné en a) : il y a entrelacement entre le log et la consultation de l'image et du texte. Image et texte sont des COMET.

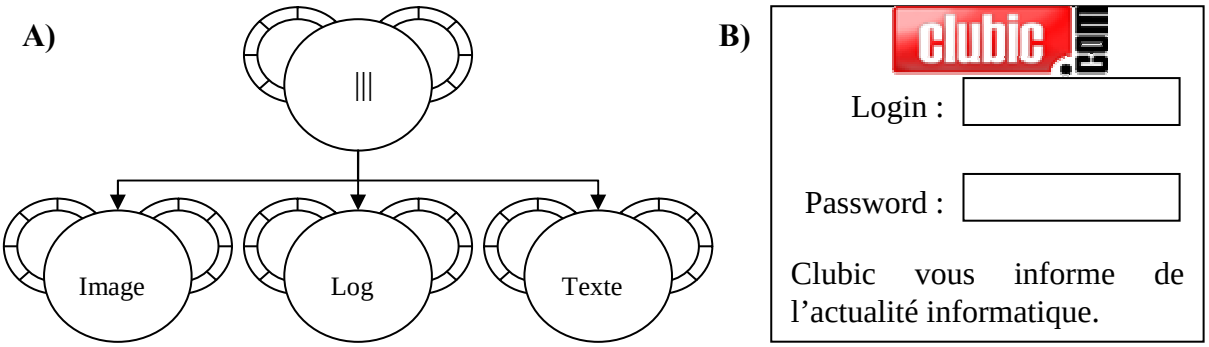


Figure III-13 : A) Entrelacement entre trois COMET : une image (titre), un log et un texte (explicatif). B) Un rendu possible.

Les graphes sont hiérarchiques : chaque élément peut contenir des COMET. Le nœud est alors dit **composite** par opposition à **atomique**. Lorsque le nœud est composite, on peut spécifier des points d'entrée et de sortie sur le graphe contenu (Figure III-14). Les points d'entrée indiquent les relations existant entre le graphe contenu et les pères du nœud composite (segment en pointillé aboutissant au rond noir). Les points de sortie indiquent les relations existant entre le graphe contenu et les fils du nœud composite (segment en pointillé sortant du rond blanc et noir). Les points d'entrée et de sortie doivent être liés à des éléments du même type (LC, LM ou PM) que l'élément composite.

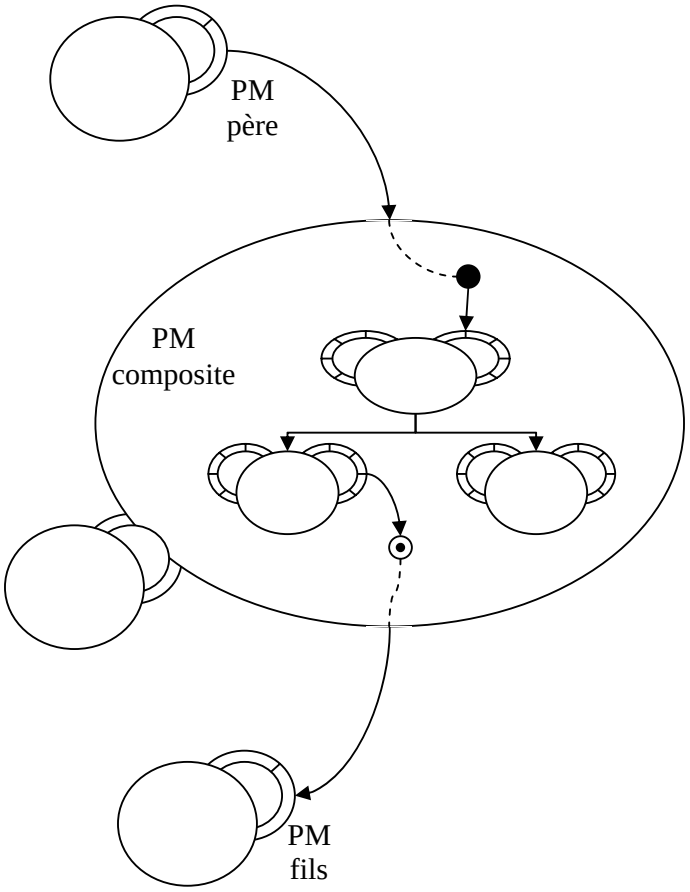


Figure III-14 : Points d'entrée et de sortie.

La Figure III-15 illustre le concept de graphe hiérarchique dans le cas d'un PM sur la COMET de Log. Le PM est celui d'une identification par couple <Identifiant, Mot de passe> avec validation explicite par l'utilisateur (cas A de la Figure III-9). Le PM contient un sous graphe composé de trois COMET : une pour l'identifiant, une autre pour le mot de passe et une pour la validation (login ou cancel). Le point d'entrée pointe ici sur un PM de présentation de la COMET Validation.

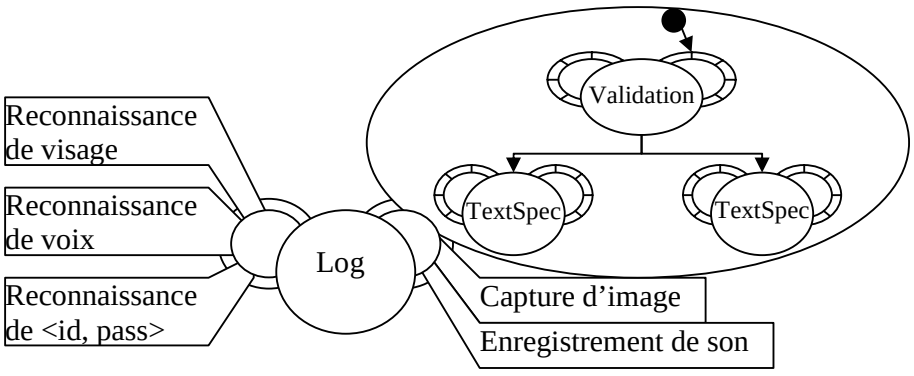


Figure III-15 : Exemple de graphe composite pour la COMET Log.

La structure de graphe hiérarchique peut être rapprochée de la notion de délégation dans les modèles à objet. La signification dépend de l'élément composite :

- Dans le cas d'un LC, le graphe contenu est une décomposition sémantique du LC.

- Dans le cas d'un LM, le graphe contenu est une décomposition spécifique de la facette de la COMET. Pour un LM de présentation, le graphe contenu exprime typiquement la décomposition en espaces de dialogue.
- Dans le cas d'un PM, le graphe contenu représente une décomposition du PM en PM plus élémentaires. Dans le cas d'un PM de présentation, le graphe contenu exprime une décomposition en interacteurs.

En pratique, le concepteur spécifie le graphe de LC. Le graphe des LM de présentation calque le graphe des LC sans entrer dans les nœuds composites. Par exemple, sur la Figure III-13, le LM de présentation de l'entrelacement a pour fils les LM de présentation de l'image, du log et du texte (comme c'est le cas au niveau des LC). Notons que seuls les LM de présentation ont une structure de graphe similaire aux LC. Cette structuration en graphe n'est pas forcément pertinente pour les autres types de LM (par exemple fonctionnels). Toutes les COMET possèdent bien des LM de présentation, mais toutes ne possèdent pas de LM fonctionnel (ou gérant des protocoles, etc.).

De même, on ne tente d'établir un graphe automatiquement que pour les PM de présentation. Le principe est de suivre le graphe des LM, mais en tenant compte cette fois de la structure hiérarchique du graphe de LM. Le graphe de LM est parcouru en passant par les graphes contenus et en veillant aux éventuelles incompatibilités entre contextes d'usage. Par exemple, dans le cas du Log, il est techniquement impossible de connecter un PM d'entrelacement HTML avec un PM de log TK. Il faut donc trouver, parmi les descendants de la COMET entrelacement, des PM de présentation compatibles du PM entrelacement HTML.

La Figure III-16 décrit l'algorithme de branchement des PM à partir d'un LM donné. L'algorithme est récursif. Il s'applique de la racine du graphe des LM à ses feuilles. A un nœud donné (un LM), l'algorithme prend deux arguments : la liste des PM instanciés dans ce LM (L_PMs – L pour List) ; les fils de ce LM (L_LMD – D pour Daughters). Pour chaque PM (PM_P), l'algorithme examine chaque LM fille (LM_F) en regardant si un de ses PM ne serait pas déjà branché au PM traité : si oui, il n'y a rien à faire. Pour ce noeud, le graphe des PM est déjà créé. Sinon, l'algorithme recherche un PM fille compatible du PM traité. Il l'instancie si nécessaire (à l'aide d'une usine) puis le branche comme fils du PM traité et réapplique récursivement l'algorithme.

```
Brancher_PMs (L_PMs : list of PM, L_LMD : list of LM)
    Pour chaque PM_P de L_PMs faire
        Pour chaque LM_F de L_LMD faire
            Si PM_P n'est pas déjà branché à un PM de LM_F alors
                Trouver PM_F parmi les PM de LM_F tel que PMF soit compatible avec PM_P
                Le cas échéant, essayer d'en usiner un
                Si PM_F existe alors | Brancher PM_F comme fils de PM_P
                                    | Brancher_PMs(PM_F, LM_F.get_daughters() )
```

Figure III-16 : Algorithme de création du graphe de PM.

L'algorithme fait appel à la notion d'usine. Une usine permet d'instancier un nouvel élément (ici un PM). Ce concept et les mécanismes associés sont intéressants pour l'extensibilité des COMET : en ajoutant de nouvelles udines, on étend le polymorphisme des présentations possibles de la COMET. Plus généralement, une usine, dans l'architecture COMET, permet d'instancier des PM reliés à n'importe quel modèle logique (LM). Ainsi, ajouter de nouvelles usines à un LM (présentation,

algorithmique, etc.) permet l'intégration à la COMET nouvelles possibilités (liées à la présentation, l'algorithmique, etc.).

L'algorithme s'appuie sur la notion de compatibilité entre PM. Cette notion s'évalue en termes des contextes d'usage requis par les PM et des branchements déjà existants. L'algorithme de détermination est décrit en Figure III-17. Son principe est le suivant : deux PM (un père PM_P et un fils potentiel PM_F) sont jugés compatibles si les trois conditions suivantes sont satisfaites :

- le père peut ajouter un fils : on peut imaginer que le nombre de fils soit borné et que le maximum soit déjà atteint. Ce pourrait être le cas pour une COMET représentant un opérateur binaire par exemple ;
- le fils peut ajouter un père : En effet, selon la technologie utilisée, certains PM n'acceptent qu'un seul père ;
- les contextes d'usage sont compatibles. Deux contextes d'usage (un père cou_P et un fils potentiel cou_F) sont dits compatibles si leurs plates-formes matérielles et logicielles le sont. D'autres critères pourraient être considérés. Cette notion est révisable. Elle dépend fortement de la modélisation du contexte d'usage adoptée.

```
Compatibles_P_F (PM_P, PM_F : PM)
  Si PM_F.nb_max_mothers() = PM_F.nb_mothers() alors → FAUX
  Si PM_P.nb_max_daughters() = PM_P.nb_daughters() alors → FAUX
  → Compatibles_P_F_CoU (PM_P.get_cou(), PM_F.get_cou())

Compatibles_P_F_CoU (cou_P, cou_F : ContextOfUse)
  Si cou_P.get_soft_ptf() !~= cou_F.get_soft_ptf() alors → FAUX
  Si cou_P.get_hard_ptf() !~= cou_F.get_hard_ptf() alors → FAUX
  → VRAI
```

Figure III-17 : Algorithme de détermination de la compatibilité entre deux PM.

Lorsque le branchement concerne un PM composite, celui-ci veille à établir les connexions nécessaires avec son graphe contenu (Figure III-18). Dans le cas où il est branché à un père, il prend le contexte d'usage de celui-ci et établit la connexion avec son graphe contenu via ses racines (nested_roots). Dans le cas où il est branché à un fils, il établit le branchement avec les feuilles du graphe contenu (nested_leaf).

```
PM_Composite :: Connecter_pere (PM_P : PM)
        Brancher_PMs(PM_P, this.get_nested_roots_L_LM() );
        Mettre à jour nested_leafs avec les PM correspondants aux points de sortie

PM_ Composite :: Connecter_fils (PM_F : PM, position : int)
        soit PM = this.get_nested_leafs()[min(position, this.get_nb_nested_leafs())]
        PM.Connecter_fils (PM_F)
```

Figure III-18 : Méthodes de connection de PM.

Le style COMET étant défini, je propose dans la section suivante des COMET d'utilité publique.

II.1.B. COMET d'utilité publique

Les COMET d'utilité publique correspondent à des tâches utilisateur ou opérateurs entre tâches classiques. Pour les identifier, on peut s'appuyer sur des taxonomies de tâches. Mon objectif ici n'est pas l'exhaustivité. Je veux juste montrer sur des exemples simples l'application du style COMET. Les API des COMET sont partiellement décrites. La section suivante montre comment étendre une BàO pour créer des COMET métier.

Root

Une COMET Root (racine) est racine d'un graphe de LC. Sa raison d'être est liée aux PM de présentation : les PM de la COMET root accueilleront les PM des COMET filles. Par exemple, en HTML, il est nécessaire de créer une page contenant les balises <head> et <body>. C'est exactement le rôle que joue le PM HTML de la COMET Root. De même, pour les autres technologies (Tk, S207 ou B207, mes boîtes à outils), un point d'ancrage est toujours nécessaire. Ce point d'ancrage est amorcé par les PM de la COMET Root (Figure III-19). En d'autres termes, d'un point de vue présentationnel, la COMET Root crée l'espace de dialogue dans lequel sont placés tous les descendants.

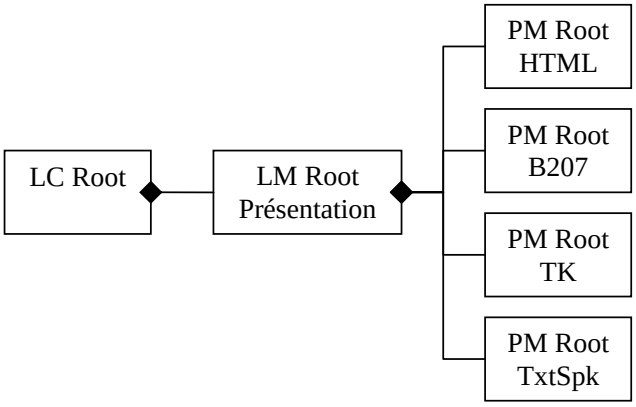


Figure III-19 : COMET Root. Plusieurs PM sont définis suivant les technologies utilisées.

Container

La COMET Container (conteneur) exprime la notion de groupement sémantique, c'est-à-dire de tâche utilisateur atteinte si les tâches filles le sont et ceci dans n'importe quel ordre, et possiblement en émiettant les sous-tâches. Du point de vue de l'AUI, le container est un espace de dialogue dans lequel les fils directs (mais pas nécessairement les descendants en général) sont tous uniformément accessibles (ils sont tous accessibles ou bien aucun ne l'est). Du point de vue de la CUI, le container peut être représenté de différentes manières : fenêtre, canevas, repère dans un espace, canvas déroulable/enroulable, etc. La diversité des représentations dépend des technologies utilisées.

Activator

La COMET Activator permet à l'utilisateur de déclencher une action. Elle introduit une API sémantique en ce sens (Figure III-20).

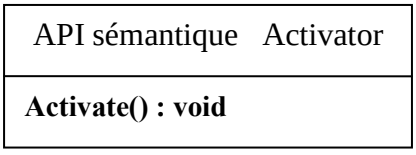


Figure III-20 : API sémantique de la COMET Activator.

Du point de vue de l'AUI, la COMET Activator est un espace de dialogue. Du point de vue de la CUI, la COMET peut être un bouton, une forme spécifique à dessiner dans une zone, une commande vocale, etc.

Interleaving

La COMET Interleaving (entrelacement) exprime l'opérateur entre tâches du même nom. La tâche est atteinte si les tâches filles le sont atteintes et ceci dans n'importe quel ordre. La différence avec le Container se manifeste au niveau de l'AUI. Du point de vue de l'AUI, la COMET Interleaving est un espace de dialogue dans lequel les filles peuvent être directement accessibles ou non. Des navigations peuvent exister entre les filles (cas des onglets typiquement) ou pas (rangement des filles en colonnes par exemple). Du point de vue de la CUI, un moyen de réaliser l'entrelacement est de s'appuyer sur la structure de graphe hiérarchique. Pour chaque COMET entrelacée (COMET 1 et COMET 2 sur la Figure III-21), on prévoit un container placé dans le PM de la COMET Interleaving. (Figure III-21-a). Le rôle de ce PM est alors de placer ces containeurs (Figure III-21-b). Cette méthode permet de combiner les PM de l'entrelacement (ligne, colonne, fleur, etc.) avec les PM des Container (fenêtre, canevas, etc.). L'ajout de nouvelles présentations de conteneurs enrichira automatiquement la présentation de l'entrelacement.

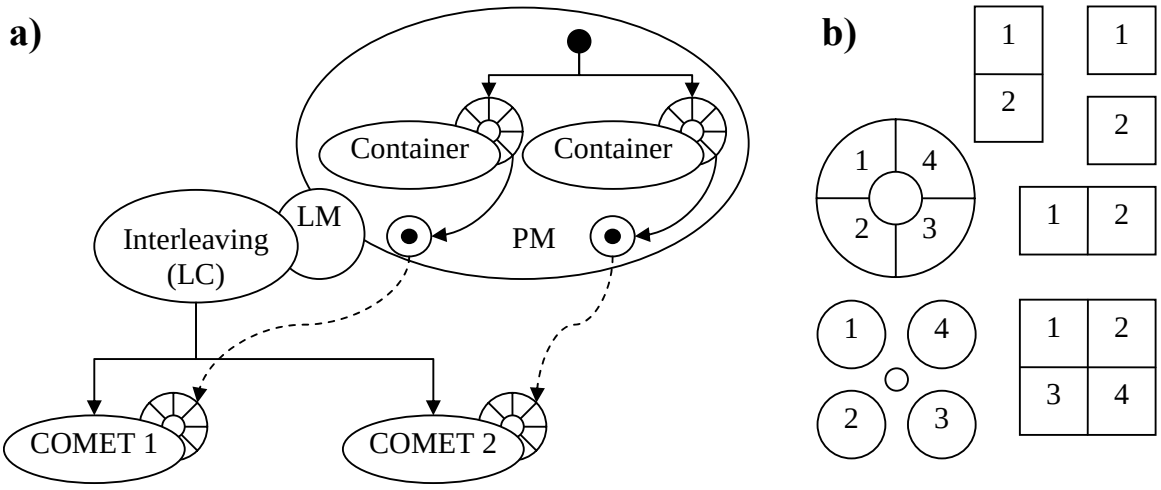


Figure III-21 : a) COMET Interleaving. b). Le PM de l'Interleaving gère la présentation et l'affichage des containers.

Bien entendu, d'autres CUI d'entrelacements sont possibles, notamment celles qui permettent de ne pas afficher tous les éléments entrelacés en même temps. (Figure III-22).

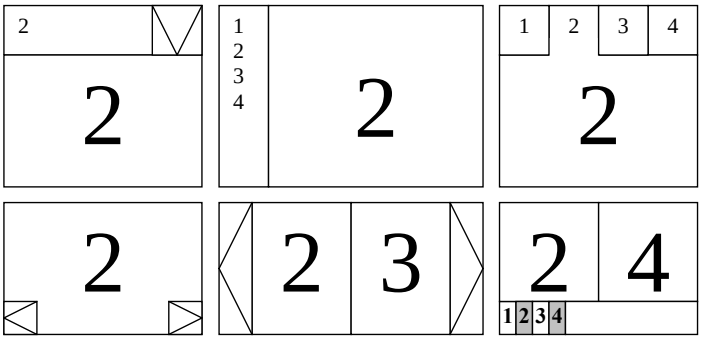


Figure III-22 : Exemples d'entrelacements permettant de ne pas afficher tous les espaces en même temps.

Un concepteur pourrait vouloir combiner différents types d'entrelacements. Prenons l'exemple du graphe de LC de la Figure III-23-a qui modélise un entrelacement entre une jauge, un indicateur CPU et un calendrier. Supposons que le concepteur veuille qu'à tout moment la jauge soit perceptible (Figure III-23-b). Dans ce cas, il pourrait s'appuyer sur la structure de graphe hiérarchique pour définir un nouveau PM composite (PM de la Figure III-23-a).

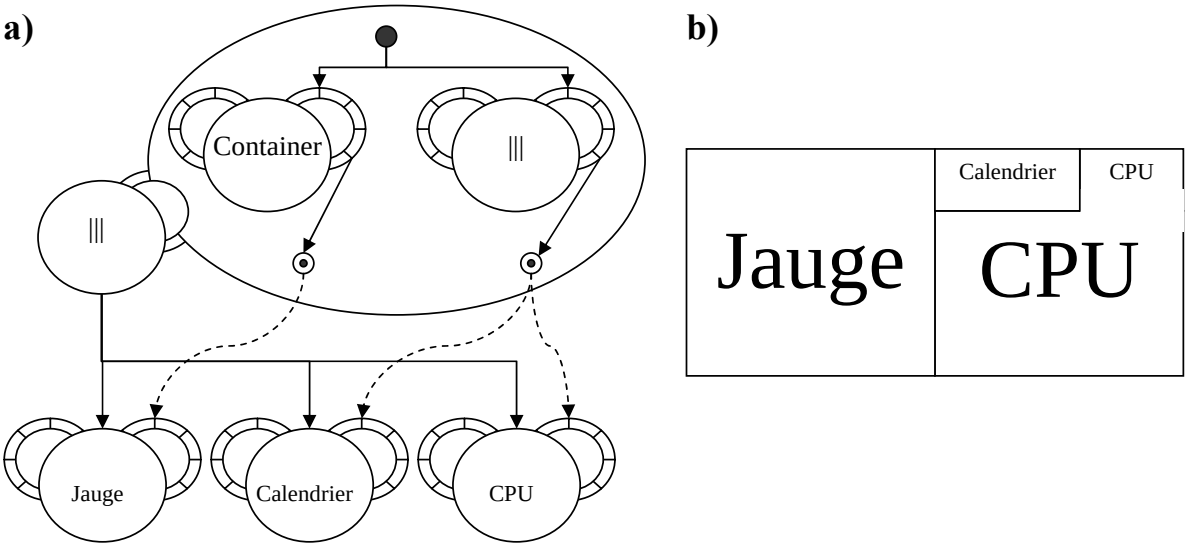


Figure III-23 : a) Redéfinition d'un PM d'entrelacement à partir d'un container et d'un entrelacement. La structure du graphe n'est pas changée aux niveaux LC et LM. b) Un exemple de rendu.

Marker

La COMET Marker (marqueur) exprime la notion de marque booléenne (Figure III-24). Par exemple, les boutons radio et les cases à cocher sont des marqueurs.

La COMET Marker gère l'état de sélection. La COMET Marker marque ses fils. La tâche de la COMET est atteinte si la marque est spécifiée et si les tâches des fils sont atteintes. Du point de vue de l'AUI, le marker est un espace de dialogue contenant les sous-espaces correspondant à ses fils. Du point de vue de la CUI, le marker peut être représenté de différentes manières (bouton radio, case à cocher, rectangle englobant, mise en surbrillance, etc.) : la diversité des représentations dépend des technologies utilisées.

API sémantique Marker
get_mark() : bool **set_mark(bool) : void**

Figure III-24 : API sémantique de la COMET Marker.

Choice

La COMET Choice (choix) exprime la notion de choix discret, c'est-à-dire d'un choix parmi un ensemble fini de possibilités. Les choix possibles sont incarnés par les fils de la COMET Choice. Une API sémantique permet la définition des nombres de choix minimum et maximum et la manipulation des éléments choisis (Figure III-25). La tâche de Choice est atteinte si le nombre d'éléments choisis est compris entre le minimum et le maximum et si les tâches des éléments choisis sont atteintes.

API sémantique Choice
get_nb_min_choices() : int **set_nb_min_choices(int) : void** **get_nb_max_choices() : int** **set_nb_max_choices(int) : void** **get_choices() : list of COMET** **set_choices(list of COMET) : void** **get_chosen() : list of COMET** **set_chosen(list of COMET) : bool**

Figure III-25 : API sémantique de la COMET Choice.

Du point de vue de l'AUI, le Choice est un espace de dialogue dans lequel les fils s'insèrent. Du point de vue de la CUI, une façon de réaliser le choix est de s'appuyer sur un entrelacement de marqueurs (Figure III-26). Cette solution a l'avantage de reposer sur des COMET existantes.

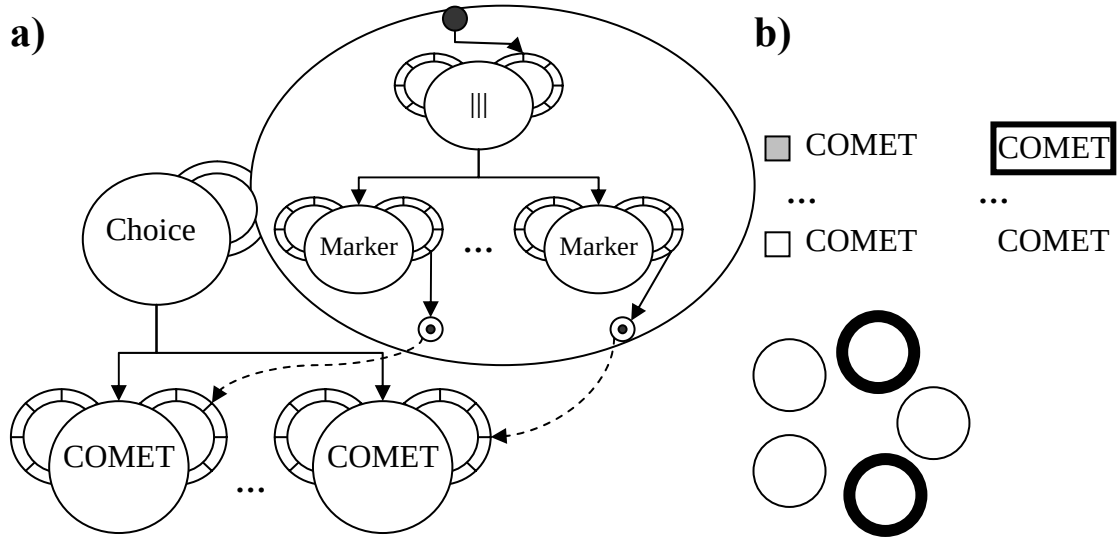


Figure III-26 : a) PM de Choice s'appuyant sur une décomposition de type Entrelacement/Marqueurs. b) Des rendus possibles.

Une autre gamme de CUI est le choix par accumulateur. Au lieu de marquer les choix faits, on les sépare géographiquement des autres (Figure III-27).

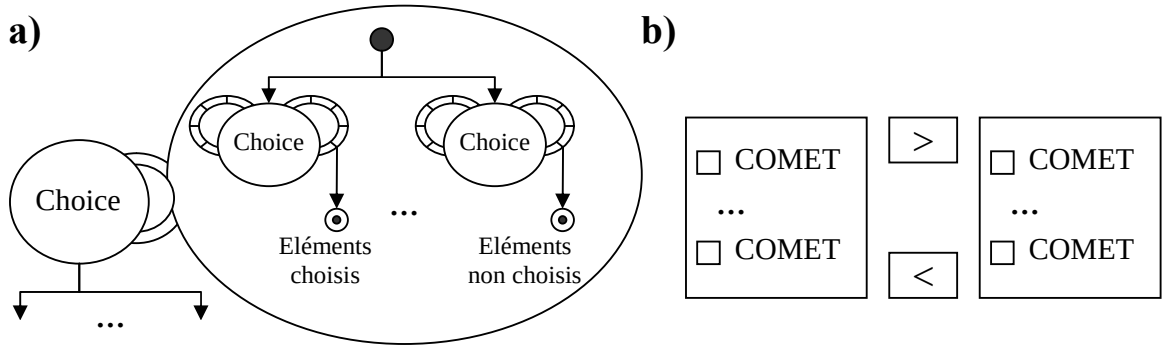


Figure III-27 : a) PM de Choice s'appuyant sur une décomposition de type Accumulateur. b) Rendu classique.

Il reste bien sûr possible au concepteur de redéfinir des PM qui lui sont spécifiques.

NChoice

La COMET NChoice (choix d'entier) exprime la notion de choix d'un entier dans un intervalle donné. Une API sémantique est définie pour permettre la spécification et la manipulation de l'intervalle et du pas (step) (Figure III-28).

API sémantique NChoice
get_b_inf() : int **set_b_inf(int) : void** **get_b_sup() : int** **set_b_sup(int) : void** **get_step() : int** **set_step(int) : void** **get_val() : int** **set_val(int) : void**

Figure III-28 : API sémantique de la COMET NChoice.

La tâche de NChoice est atteinte si un entier a été spécifié dans l'intervalle donné. La COMET NChoice n'a pas de fils. Du point de vue de l'AUI, la COMET NChoice est un espace de dialogue. Du point de vue de la CUI, NChoice peut être représenté par un potentiomètre, une jauge, un compteur, un champ texte, etc.

Image

La COMET Image permet de représenter une image. Elle introduit une API sémantique permettant de charger une image à partir d'une localisation (fichier). La tâche de la COMET est toujours atteinte. Du point de vue de l'AUI, c'est un espace de dialogue. Du point de vue de la CUI, c'est une image ou un texte (le nom de l'image). L'intérêt des différents PM de présentation est de permettre le rendu de l'image dans différentes BàO (HTML, TK, OpenGL, etc…).

Hierarchy

La COMET Hierarchy exprime la notion de hiérarchie. En terme de graphe, cette COMET permet de représenter des DAG (Graphes Acycliques Dirigés) et de naviguer dans ces DAG. La tâche de la COMET est toujours atteinte. Elle introduit une API sémantique permettant de construire le DAG à partir d'une racine (root) et d'une commande (command) permettant d'obtenir la liste des fils d'un nœud (Figure III-29). Cette COMET n'a pas de fils.

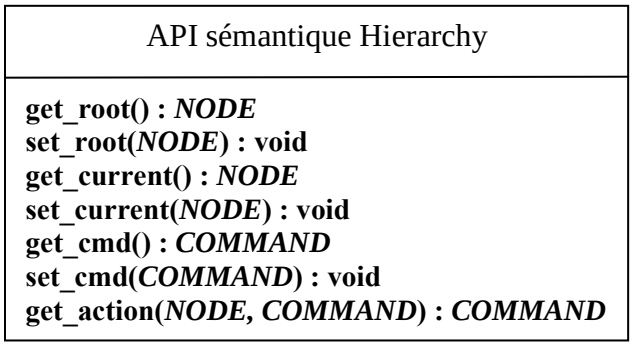


Figure III-29 : API sémantique de la COMET Hierarchy.

Du point de vue de l'AUI, la COMET Hierarchy est un espace de dialogue. Du point de vue de la CUI, elle peut être représentée par un graphe, une liste arborescente, un menu, etc.

TextSpecifyer

La COMET TextSpecifyer permet de spécifier un texte. Sa tâche est toujours atteinte, sauf si une expression régulière lui a été associée, auquel cas la tâche n'est atteinte que si le texte spécifié est conforme à cette expression. Elle introduit une API sémantique permettant de spécifier et manipuler le texte (Figure III-30).

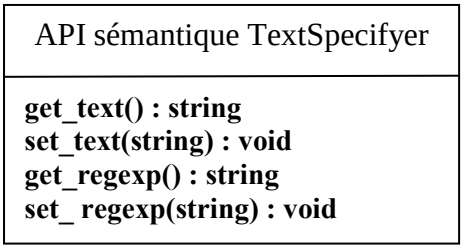


Figure III-30 : API sémantique de la COMET TextSpecifyer.

Du point de vue de l'AUI, la COMET TextSpecifyer est un espace de dialogue. Du point de vue de la CUI, elle peut être un champ texte, une zone de texte, une zone de reconnaissance manuscrite, un module de reconnaissance vocale, etc.

Clock

La COMET Clock permet de consulter l'heure courante. Sa tâche est toujours atteinte. Elle introduit une API sémantique permettant d'obtenir l'heure en terme du nombre de millisecondes écoulées depuis 1972 (Figure III-31). La COMET Clock n'a pas de fils.

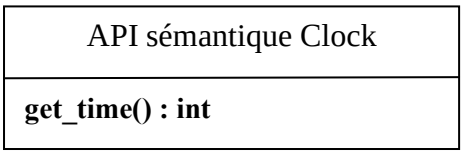


Figure III-31 : API sémantique de la COMET Clock.

Du point de vue de l'AUI, la COMET Clock est un espace de dialogue. Du point de vue de la CUI, plusieurs présentations sont possibles, notamment des présentations analogiques ou numériques (Figure III-32).

La COMET Clock dispose d'un LM fonctionnel chargé d'obtenir l'heure courante. Plusieurs PM fonctionnels sont possibles. La Figure III-32 en présente deux : l'un détermine l'heure à partir de l'horloge interne de l'ordinateur ; l'autre accède à une horloge atomique via Internet.

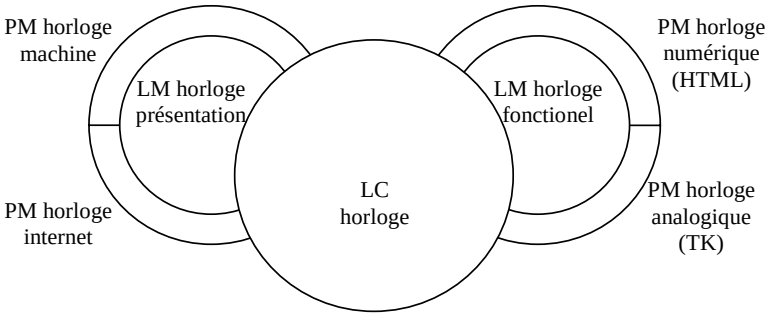


Figure III-32 : Plusieurs PM de rendu et fonctionnels peuvent être utilisés dans la COMET Clock.

Le concepteur peut bien sûr ajouter à loisir des PM fonctionnels ou de présentation.

Validation

La COMET Validation exprime la notion de confirmation. Elle a pour fils les COMET dont il s'agit de valider les actions utilisateur. Sa tâche est de permettre à l'utilisateur de confirmer ou d'infirmer les actions faites sur ses fils (ex : valider ou annuler les données d'un formulaire). Sa tâche est accomplie dès que la confirmation ou l'infirmation est faite. Elle introduit une API sémantique en ce sens (Figure III-33).

API sémantique Validation
Confirm() : void **Cancel() : void**

Figure III-33 : API sémantique de la COMET Validation.

Du point de vue de l'AUI, la COMET Validation est un espace de dialogue contenant ses fils. Du point de vue de la CUI, une façon de réaliser la COMET Validation est de d'utiliser la structure de graphe hiérarchique en s'appuyant sur des COMET Container et Activator. Le point d'entrée du graphe contenu est un PM du noeud racine du graphe contenu. Le point de sortie est un PM du conteneur de gauche (Figure III-34-a). C'est sous ce point de sortie que les fils de la COMET Validation se brancheront (« … » dans la Figure III-34-b).

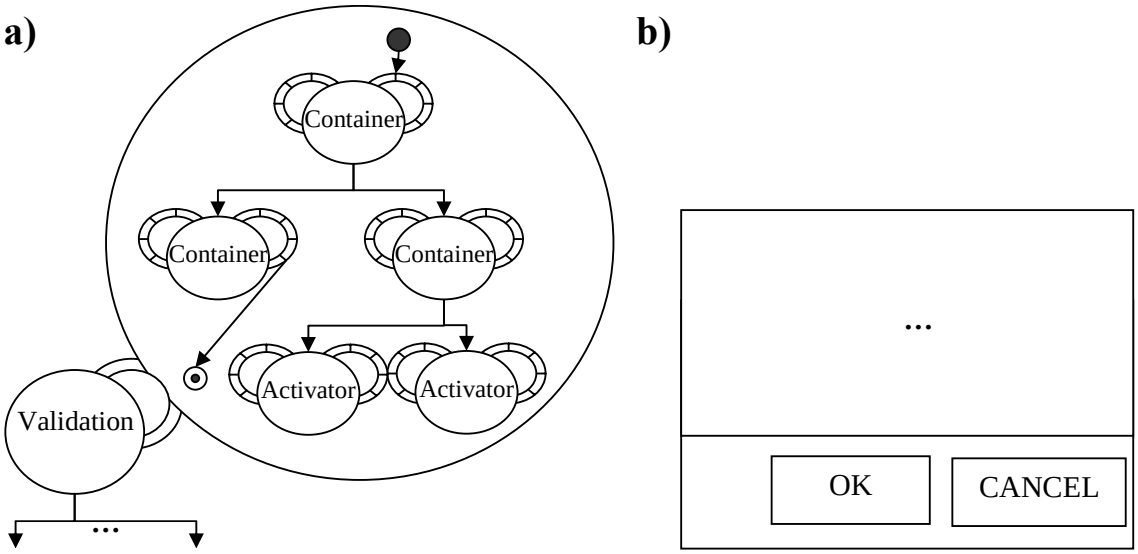


Figure III-34 : a) Exemple de PM réalisant la COMET Validation avec des Activator (OK, CANCEL) et des Container. b) Une représentation possible.

Le concepteur peut bien entendu personnaliser à façon.

Sequence

La COMET Sequence exprime la notion d'ordre temporel dans la réalisation de tâches. Elle prend pour fils une liste de COMET dont la réalisation, dans cet ordre, est nécessaire pour que la tâche de la COMET soit atteinte. La COMET Séquence introduit une API sémantique permettant de connaître parmi les fils de la COMET la tâche courante.

Du point de vue de l'AUI, la COMET Sequence est un espace de dialogue. Du point de vue de la CUI, elle peut être représentée de nombreuses façons : séquence temporelle (les fils sont présentés les uns après les autres), représentation de tous les fils mais un seul est manipulable à un moment donné (les autres sont inactifs, grisés par exemple). La Figure III-35-a montre un exemple de PM composite pouvant donner lieu au rendu de la Figure III-35-b. D'autres types de rendus, comme par exemple « tous les fils grisés sauf le courant », nécessitent d'autres structurations du graphe contenu.

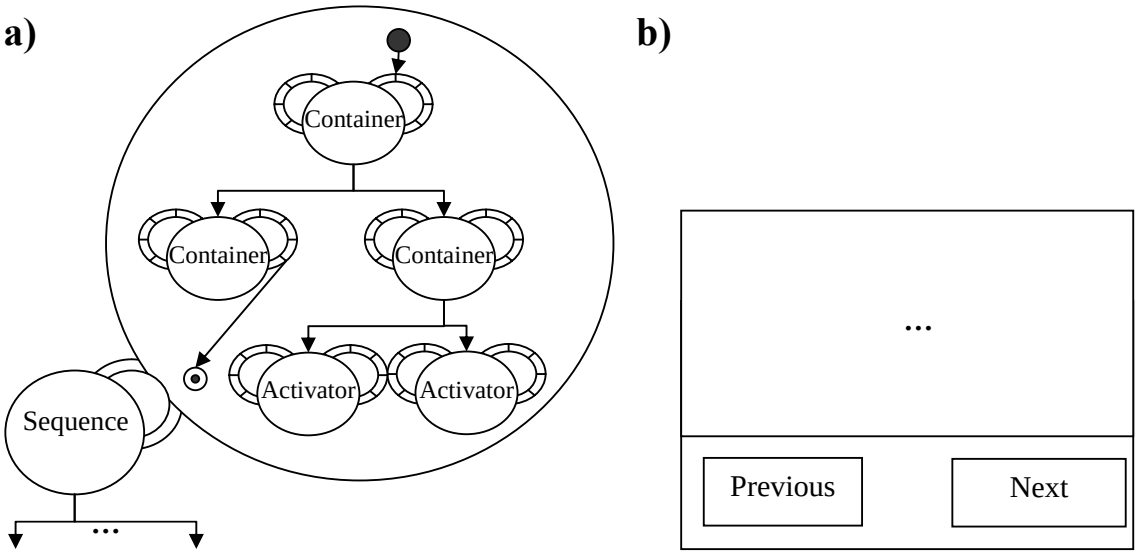


Figure III-35 : a) Un PM de séquence. La structure du graphe contenu est similaire à celle présentée pour la COMET Validation dans la Figure III-34. b) Une représentation possible de ce PM.

OR

La COMET OR représente la notion de choix pour la réalisation d'une tâche. La COMET OR prend une liste de fils. Chaque fils décrit une façon de réaliser fonctionnellement une tâche. Cette tâche est la même pour tous les fils. Seule la façon de la décomposer change. La tâche de la COMET OR est atteinte si un des fils est mené à bien.

Du point de vue de l'AUI, la COMET OR est un espace de dialogue dans lequel s'insèrent les fils. Du point de vue de la CUI, elle peut être représentée de diverses façons : menu donnant accès aux fils, onglets, etc.

Inspector

La COMET Inspector [41-42 Demeure 2006] représente la tâche d'inspection d'un graphe de COMET. C'est en quelque sorte une Méta-COMET. La COMET Inspector ne prend pas de fils. Elle introduit une API sémantique (Figure III-36) permettant de manipuler un graphe de COMET. Il est possible de charger/sauver des styles, d'ajouter de nouvelles règles de style ou d'en supprimer. Il est possible d'accéder et de modifier l'élément racine et l'élément courant (qui peuvent être un LC, un LM ou un PM). L'architecture logicielle de la COMET correspondant à l'élément courant est

représentée. Enfin, des opérations sont applicables sur l'élément courant. On peut lui ajouter des fils, en supprimer ou encore substituer des PM qui lui sont liés.

API sémantique Inspector
get_root() : CometElement **set_root(CometElement) : void** **get_current() : CometElement** **set_current(CometElement) : void** **Load_style(string) : bool** **Save_style(string) : bool** **Apply_style(CometElement) : void** **set_GDD_functions(string) : void** **get_GDD_functions() : string** **Add_rule(string, string) : void** **Sub_rules(set of string) : void** **Substitute_PM(PM, PM) : bool** **Add_daughter_to_current(CometElement) : bool** **Sub_daughter_to_current(CometElement) : bool**

Figure III-36 : API sémantique de la COMET Inspector.

La COMET Inspector est fonctionnellement décomposée en quatre parties (Figure III-37) : gestion des styles (CometStyler, non décrite dans la thèse pour raison de concision), gestion du graphe de COMET (Hierarchy), vue conceptuelle détaillée de l'élément courant (CometViewer, non décrite dans la thèse) et opérations applicables (CometOperations, non décrite dans la thèse). Chaque modification dans l'une des parties est propagée aux autres. Par exemple, si la racine est changée, alors CometViewer et les opérations sont modifiées en conséquence. Si on change l'élément courant (avec CometViewer par exemple), alors l'élément courant devient la nouvelle racine de la hiérarchie.

Du point de vue de l'AUI, la COMET Inspector est un espace de dialogue contenant quatre sous espaces (un pour chaque partie). Du point de vue de la CUI, une façon de réaliser la COMET Inspector est d'utiliser la structure de graphe hiérarchique contenue dans le LC (PM de la Figure III-37). La Figure III-38 donne une représentation possible de la COMET Inspector.

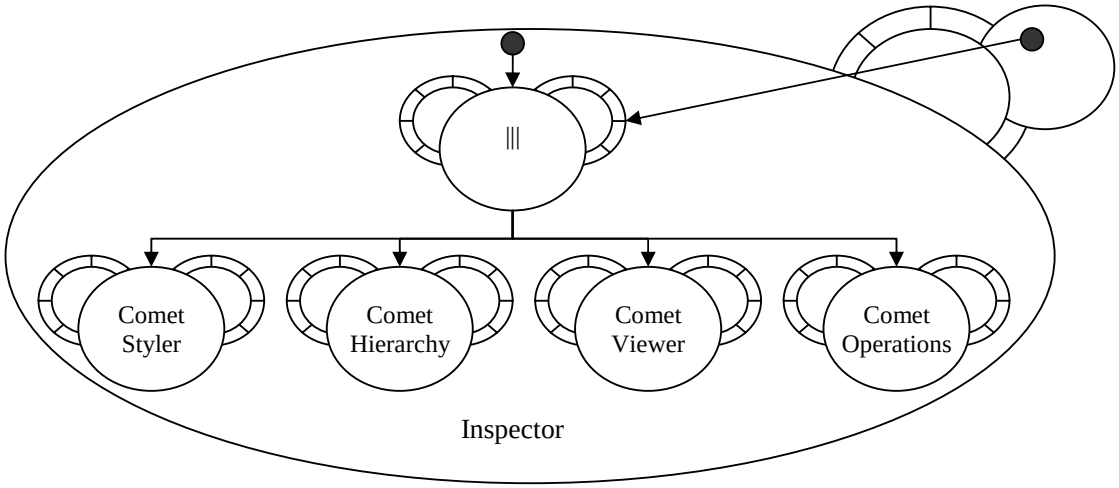


Figure III-37 : COMET Inspector.

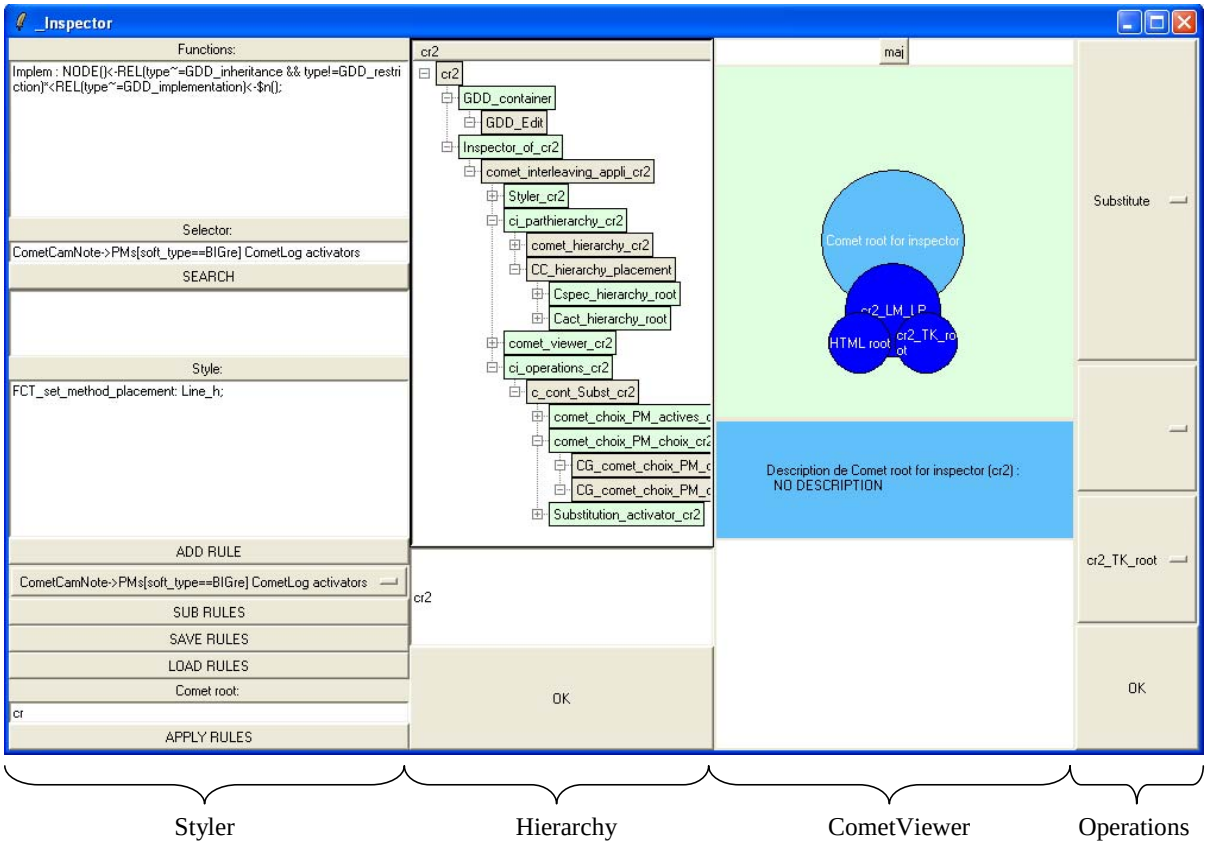


Figure III-38 : Une représentation possible de la COMET Inspector. Les quatre parties (Styler, Hierarchy, CometViewer et Operations) sont placées en ligne.

Le concepteur peut étendre ces COMET d'utilité publique par la création de COMET métier. C'est l'objet de la section suivante.

II.1.C. Conception d'une COMET

Concevoir une COMET, c'est concevoir son LC, ses LM et ses PM. Il n'y a aucune contrainte de « grain » : une COMET est un système interactif, quelle que soit l'ampleur de la tâche considérée. D'un point de vue méthodologique, j'indique ici les étapes à suivre et les questions à se poser dans la conception d'une COMET. Une COMET simple de messagerie instantanée est utilisée comme illustration (type [159 Trillian]) :

1. Définir le LC et son API sémantique. L'API définit le langage qui sera partagé par toutes les facettes. Il faut donc se poser la question de « quelle est la sémantique de la COMET ? Quelle tâche, quel service doit-elle rendre ? ». Dans l'exemple de la messagerie, la COMET doit permettre l'échange de messages avec d'autres utilisateurs. Elle doit donc permettre de consulter l'historique des messages, d'en saisir de nouveaux et de les envoyer. L'API sémantique doit donc inclure les trois fonctions suivantes : « **void send (string msg); string get_historic(); void add_to_historic(string h);** ».
2. Définir les LM. La question à se poser est « quels sont les aspects particuliers nécessaires à la réalisation de la tâche de la COMET ? ». Dans l'exemple de la messagerie, il faut, par exemple, gérer l'interaction avec l'utilisateur : un LM est dédié à cette interaction. Chaque LM doit implémenter l'API sémantique du LC. Pour chaque LM, il faudra éventuellement étendre l'API pour prendre en compte les problèmes spécifiques liés au LM. Par exemple, le LM de présentation pourra

gérer les émoticons (Figure III-39). D'un point de vue fonctionnel, il est utile que la COMET puisse dialoguer avec d'autres logiciels de messagerie instantanée, comme MSN, YAHOO ou AIM. Un LM dédié à la gestion des protocoles réseaux est nécessaire. Il étendra l'API avec une méthode gérant la réception de nouveaux messages provenant du réseau (Figure III-39).

3. Décider de l'encapsulation de graphes pour tous ces éléments : dans l'ordre, LC puis LM. Cette question revient à se demander si l'élément considéré peut ou non embarquer un graphe de COMET auquel il pourra déléguer sa tâche. Dans l'exemple de la messagerie, le LC pourrait être sémantiquement décomposé en une COMET de spécification de texte (message à envoyer) et une COMET de rendu de texte (historique des messages). Les choix faits à un niveau (par exemple LC) peuvent influencer les autres niveaux (LM et PM).

4. Définir les PM pour chaque LM. Chaque PM doit implémenter l'API sémantique de son LM. Si le LM ou le LC correspondant au PM sont composite, le PM peut choisir de s'appuyer sur les graphes contenus. Par exemple le PM pourrait faire appel à la COMET Text pour rendre l'historique. Une autre stratégie peut être l'utilisation de COMET d'utilité publique. Le PM est alors composite, fait de COMET d'utilité publique. La question à se poser est alors : « Y a-t-il des PM particuliers qui ne puissent être obtenus par décomposition en COMET d'utilité publique ? ». Dans l'exemple de la messagerie, d'un point de vue fonctionnel, il est nécessaire d'implémenter les PM gérant les différents protocoles réseaux. En revanche, en terme de présentation, les PM peuvent être implémentés comme composites de COMET Text et TextSpécifyer. Des présentations originales pourraient être ajoutées, par exemple des animations en OpenGL donnant alors lieu à des PM spécifiques.

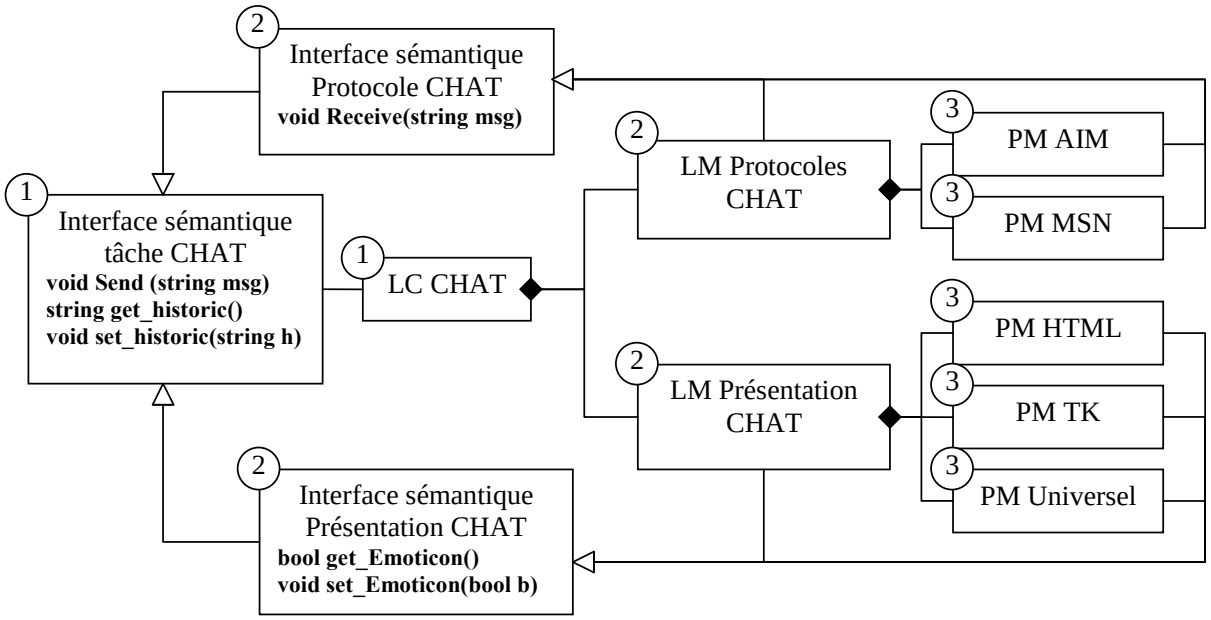


Figure III-39 : COMET de messagerie instantanée (dite CHAT). Les étapes de conception sont numérotées.

Les implémentations des méthodes de l'API sémantique doivent assurer la cohérence entre les éléments du modèle. Ainsi, le LC doit assurer la cohérence des modèles logiques (LM). Les LM doivent assurer la cohérence des PM qui leur sont liés. Les PM doivent quant à eux fournir un moyen de réaliser concrètement les méthodes en s'appuyant sur des technologies particulières. Les PM composites sont alors

particulièrement utiles car ils permettent de définir des présentations de COMET en se basant sur des COMET d'utilité publique. L'avantage de cette approche est de pouvoir proposer des rendus dans une large variété de technologies sans avoir pour autant à spécifier, pour chaque COMET, les PM dans chacune de ces technologies. Seules les COMET d'utilité publique devront être réalisées pour chaque technologie. L'inconvénient de cette approche est, en corollaire, la relative pauvreté des IHM produites dans la mesure où elles se basent toutes sur un même ensemble de rendus.

Reprenons l'exemple de la COMET de messagerie instantanée et supposons que nous la structurions comme indiqué dans la Figure III-40. Dans cette décomposition, le LC de la COMET délègue la gestion de l'historique à une COMET de représentation de texte (Text). Il délègue aussi la gestion du message à envoyer à une COMET de spécification de texte (Text Spec). Un des PM de présentation est un composite. Il utilise les COMET contenues dans le LC pour construire sa propre présentation. Un autre PM, atomique cette fois, propose sa propre présentation indépendamment des COMET contenues dans le LC.

D'un point de vue fonctionnel, plusieurs protocoles de communication de messagerie peuvent être proposés, simultanément ou pas (Figure III-40).

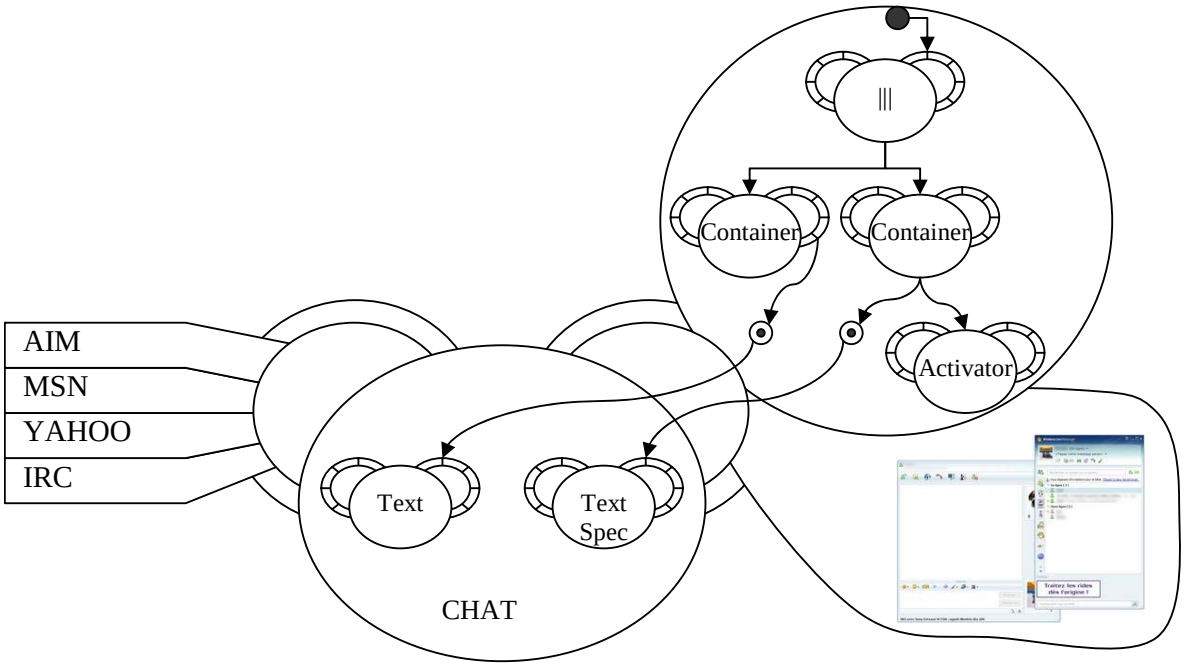


Figure III-40 : COMET de messagerie instantanée (CHAT).

La section suivante montre quelques forces du style COMET qui dépassent les objectifs initiaux.

II.1.D. Potentialités du modèle

J'identifie trois atouts supplémentaires par rapport aux objectifs initiaux. Ils sont relatifs à la dynamicité du modèle, au traitement de la multimodalité et de l'ergonomie en particulier par un mécanisme général d'encapsulation.

La multimodalité avec les COMET

Par modalité, j'entends un couple <langage d'interaction, dispositif (physique ou numérique)> [27 Coutaz 1995]. Par exemple, un bouton pouvant être activé en cliquant dessus avec un pointeur (numérique) a une modalité d'entrée <bouton cliquable,

pointeur numérique>. Le même bouton, activable par la touche ENTREE du clavier, a une modalité d'entrée <bouton cliquable, touche ENTREE du clavier>. Différentes modalités de sortie sont imaginables : <bouton rond OpenGL, repère X> ou <bouton rectangulaire HTML, Navigateur WEB FireFox>.

Par multimodalité, j'entends la combinaison de modalités. Les propriétés CARE sont classiques pour raisonner sur ces combinaisons. CARE signifie Complémentarité, Assignation, Redondance, Equivalence. Je m'intéresse à la Redondance et à l'Equivalence. Ces deux propriétés combinent des modalités en entrée et/ou sortie avec une exigence de cohérence entre les modalités. Le style COMET est approprié pour ces deux propriétés. En revanche, je ne considère pas l'Assignation. Cette propriété est le résultat d'un usage des modalités. Je ne considère pas non plus la complémentarité. Une vraie complémentarité doit gérer les déictiques à la « mets ça là » [15 Bolt 1980], Le problème est complexe. Ce n'est pas l'objet de ma thèse. En revanche, les COMET sont appropriées pour en incarner des formes « dégradées » :

- La complémentarité en entrée pour des modalités permettant ensemble la réalisation d'une tâche élémentaire. Par exemple, la tâche Spécifier texte (tâche élémentaire) pourra être réalisée de façon complémentaire à la voix et au clavier par alternance de ces deux formes de saisie. Le style COMET couvre par construction cette forme de complémentarité. Sur la Figure III-41-a, l'utilisateur entre une partie du texte à la voix (1). La modification est propagée au LM (2). Il la renvoie aux autres PM (3). Toutes les présentations sont ainsi mises à jour avec le texte entré vocalement. En b, l'utilisateur saisit la suite du texte au clavier (1). Il est transmis au LM (2) qui le propage aux autres PM (3).

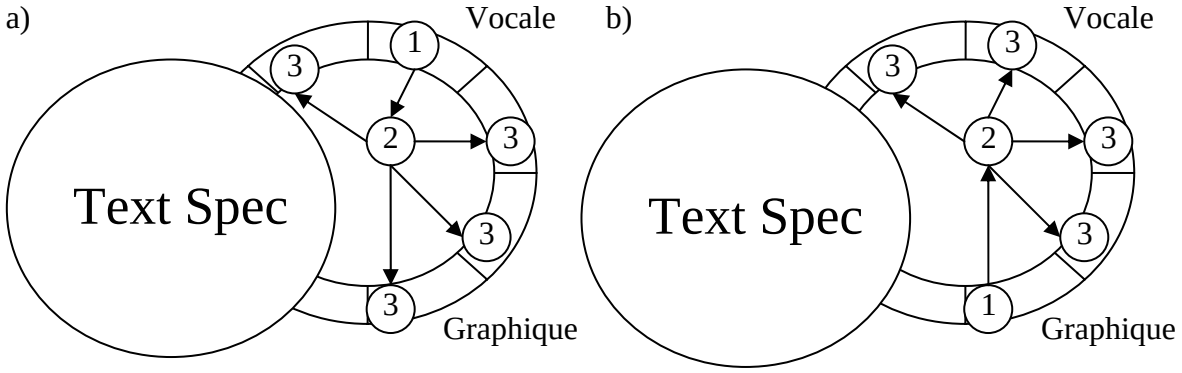


Figure III-41 : Complémentarité alternée avec les COMET.

- La complémentarité en entrée pour des tâches composées. Par exemple, positionner la caméra, c'est la déplacer et l'orienter. On peut imaginer recourir à des modalités différentes pour chacune de ces sous-tâches. La tâche composée est alors réalisée par complémentarité des modalités. Le style COMET réalise naturellement cette complémentarité. Un des PM est utilisé pour modifier la position, un autre pour modifier l'orientation.
- Enfin, la complémentarité en sortie est acquise par construction dans les COMET : le LC est chargé de la cohérence sémantique de la COMET ; les LM sont chargés de la cohérence des PM ; les PM assurent les présentations avec, en particulier, une diversité technologique dans les rendus.

Pour la Redondance et l'Equivalence, pour lesquelles il n'existe rien de base dans les COMET, je définis le langage COMET/RE. L'idée est d'associer une expression

COMET/RE aux méthodes sémantiques des LM. Ces méthodes gèrent, en effet, la cohérence de la COMET entre les PM d'une part mais aussi avec le LC. C'est donc dans les LM que doit s'ancrer le traitement de cette forme de multimodalité. L'expression COMET/RE associée à une méthode F du LM contrôle comment propager l'appel de F aux LC et PM. Dans le cas d'une redondance, la propagation sera tempérée en l'attente d'un autre événement réalisant la redondance. Dans le cas d'une équivalence, l'événement sera propagé. La Figure III-42 donne un exemple d'équivalence. Une phrase du langage COMET/RE (<E(*)>) placée dans le LM impose une équivalence pour toutes les méthodes F.

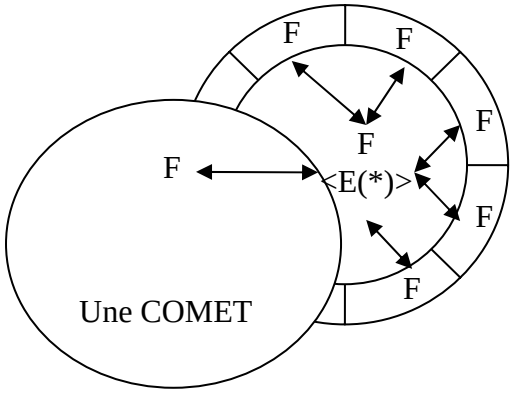


Figure III-42 : Ancrage de la multimodalité dans les LM.

La grammaire du langage est fournie dans le Tableau III-2.

Tableau III-2 : Grammaire BNF du langage COMET/RE.

Symbole	Traduction	Commentaires
EXPR	OP '(' [TPS ','] L_EXPR ')'	Un opérateur s'applique sur une liste d'expressions pour un temps TPS donné en millisecondes (optionnel).
OP	R | E	Redondance, Equivalence
L_EXPR	EXPR ',' L_EXPR | ARG	Une liste d'expressions est composée d'expressions ou bien d'une liste d'arguments
L_ARGS	ARG | ARG ',' L_ARGS	Une liste d'arguments est composée d'au moins un argument.
ARG	'*' | 'gfx' | 'vocal' | 'tactil' | ID | BAO	Un argument représente des PM. * signifie « tous les PM ». gfx/vocal/tactil signifient respectivement tous les PM graphiques/vocaux/tactiles. ID identifie un PM. BAO identifie les PM d'une certaine BàO.
ID	*string*	Un identifiant de PM
BAO	*string*	Un nom de BàO (ex HTML)

Appliquons le langage COMET/RE à la COMET de messagerie instantanée.

Tableau III-3 : Exemple d'expressions COMET/RE pour la COMET de messagerie instantanée.

Méthode et expression	Effet produit
LM de présentation, méthode **send** : E(*)	Il est suffisant qu'un des PM active sa méthode **send** pour que LC.send soit appelé, ainsi que la méthode **send** des autres PM.
LM de présentation,	Il est nécessaire que tous les PM activent leur méthode **send**

méthode **send** : R(*)	avec les mêmes paramètres pour que **LC.send** soit appelé.
LM de présentation, méthode **send** : R(E(gfx), E(vocal))	Il est nécessaire qu'au moins un PM graphique et au moins un PM vocal activent leur méthode **send** avec les mêmes paramètres pour que l'appel soit propagé.
LM de protocoles, méthode **receive** : R(1000, *)	Il est nécessaire que la COMET reçoive le même message de la part de tous les protocoles dans un intervalle de 1 seconde maximum pour que **LC.receive** soit appelé.

Encapsulation d'éléments

L'encapsulation est une technique intéressante pour enrichir des présentations, en particulier, d'un point de vue ergonomique. Prenons l'exemple très simple d'une application de gestion de présentations. Cette application oblige l'utilisateur à se logger avant de pouvoir suivre la présentation. Tout en se loggant, l'utilisateur peut par contre percevoir l'état de la présentation (pas démarré, démarré depuis N minutes, terminée). Le modèle des tâches est donné en Figure III-43.

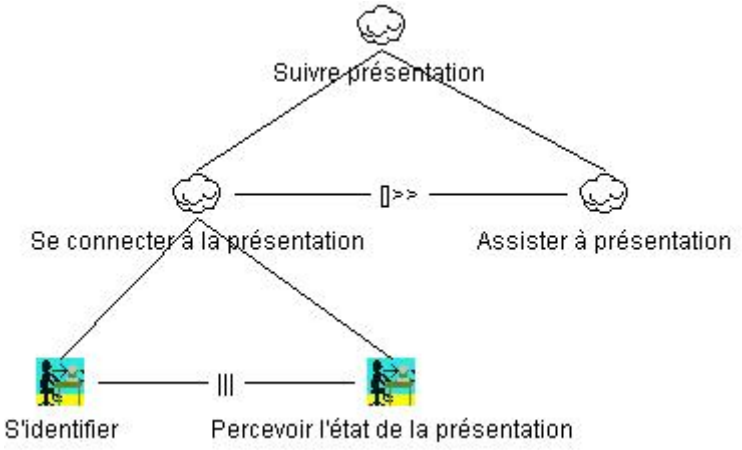


Figure III-43 : Fragment du modèle des tâches d'une application de gestion de présentation.

Une mise en oeuvre brutale avec une BàO traditionnelle (TK, HTML, etc.) donnerait, pour la tâche « Se connecter à la présentation », la Figure III-44-A : un champ texte pour le login ; un champ texte pour le mot de passe et un libellé pour l'état de la présentation. Cette IHM succincte transgresse, en particulier, le critère de Guidage / sous-critère Incitation [10 Bastien 1993] : les champs texte doivent en effet être annoncés par des libellés qui indiquent la nature, le format et les valeurs acceptables pour les informations à entrer. Ajouter ces guidages avec une BàO classique, c'est insérer, au même niveau hiérarchique, des widgets supplémentaires à vocation de guidage. Mais le lien entre ces libellés et les champs texte qu'ils annoncent n'est alors pas explicite. La modélisation correspondante est donnée en Figure III-44-B.

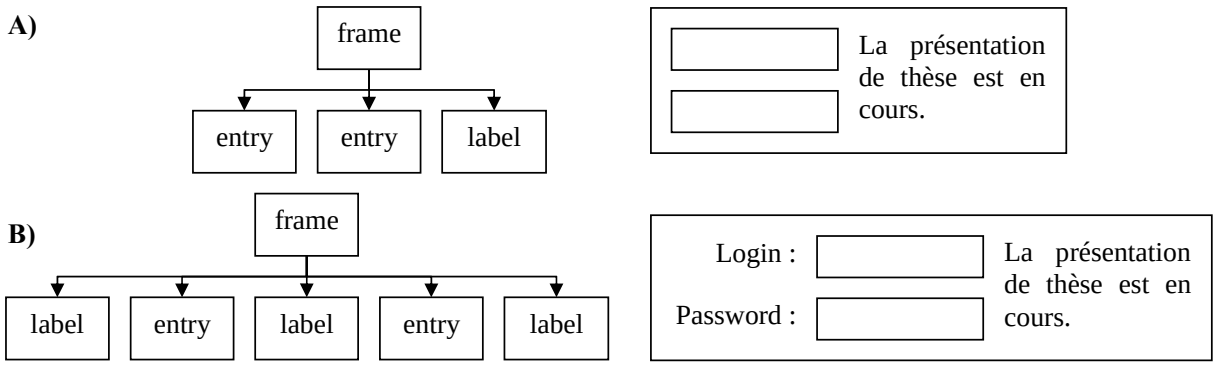


Figure III-44 : A) IHM sans guidage. B) Ajout d'informations de guidage avec des BàO "classiques".

L'inconvénient de la Figure III-44-B est la banalisation des widgets de guidage. En effet, si une adaptation a comme objectif de réduire la superficie du rendu, il est impossible, par le seul graphe de widgets, de distinguer ceux à vocation fonctionnelle (saisie de texte par exemple) des widgets de « confort ».

Grâce aux graphes hiérarchiques, un principe d'encapsulation peut être mis en œuvre avec les COMET. Ce principe est général mais règle, en particulier, la préoccupation de l'ergonomie évoquée ci-dessus. L'encapsulation permet une meilleure séparation des préoccupations dans le graphe de COMET. La Figure III-45-A présente la modélisation « brutale » en COMET de l'application de gestion de présentation. La Figure III-45-B présente la modélisation de cette même application mais avec cette fois des informations de guidage pour les champs texte. Cette modélisation donnerait par exemple le rendu de la Figure III-44-B.

L'astuce consiste à encapsuler dans les LM de présentation les informations de guidage. Le LM est composite. Il est dit Encapsulateur. Il embarque un graphe de COMET et assure la cohérence entre le LC et le LM d'origine en redirigeant les appels. Il gère un ensemble de PM universels de spécification de texte (des PM universels implémentant l'API sémantique de la COMET TextSpecifyer). La structure du graphe contenu est telle que le seul point de sortie du graphe est automatiquement placé sous le LM encapsulé. Ainsi, le LM d'origine conserve ses fils une fois encapsulé. Pour maintenir la conformité au style COMET, un LC est généré pour le LM encapsulé (à un LM correspond toujours un LC). Ce LC fait le lien avec le LM encapsulateur en redirigeant les appels (LC généré ↔ LM encapsulateur).

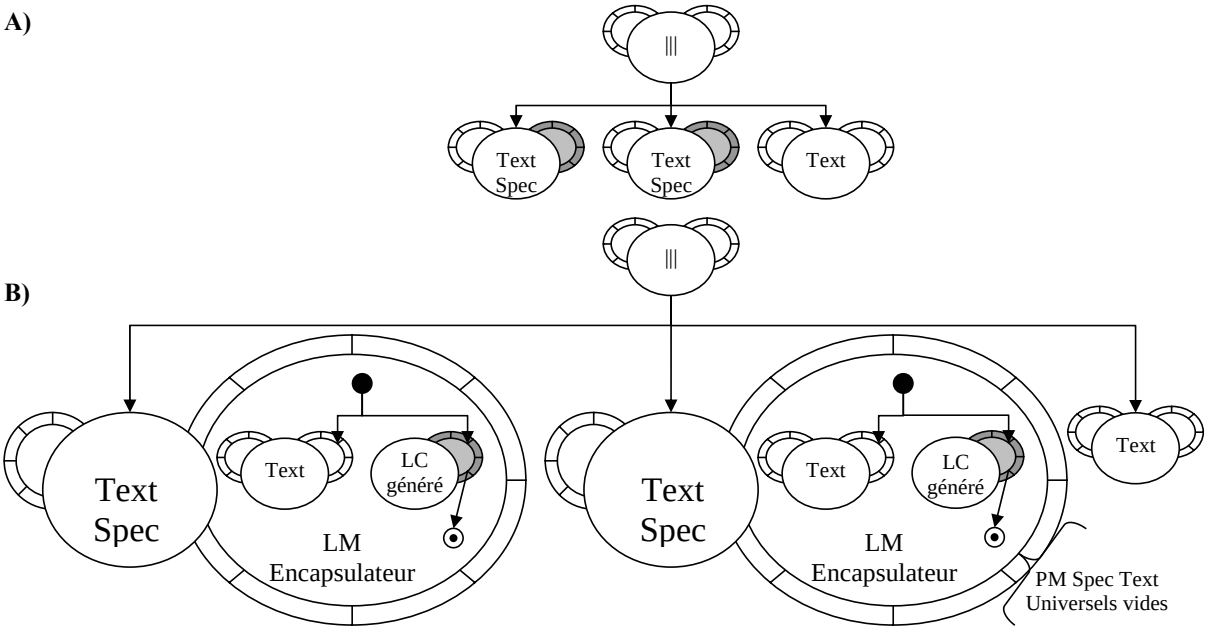


Figure III-45 : A) IHM sans guidage. B) Ajout d'informations de guidage par encapsulation.

La Figure III-45 illustre le principe d'encapsulation dans les LM. Pour l'exemple, ceci revient à dire que chaque PM de spécification de texte se verra adjoindre un PM de représentation de texte (ex : un libellé). L'avantage de cette approche est que le graphe des LC est préservé entre les Figure III-45 A et B. La sémantique de l'application est bien inchangée. Le graphe des LM reste sensiblement le même à l'exception des noeuds composites. Seule la structure du graphe des PM change, ce qui reflète bien la nature du changement opéré entre la modélisation « brutale » et la version réfléchie avec guidage.

Il est possible d'appliquer ce même principe au niveau des PM (Figure III-46). La différence est que la modification sera localisée au niveau d'un PM sans impacter les autres PM. Sur l'exemple, ceci reviendrait à estimer que l'ajout de guidage n'est pas opportun pour tous les PM. Le PM ciblé est alors détaché de son LM initial. Il est encapsulé dans un PM encapsulateur. Ce PM encapsulateur est lié à un LM et un LC générés de façon à respecter le style COMET.

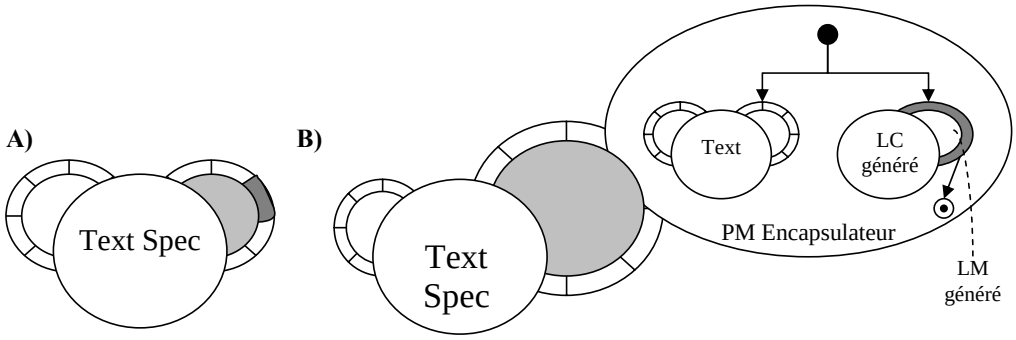


Figure III-46 : A) La COMET originale de spécification de texte. B) La COMET après encapsulation du PM grisé.

II.1.E. Conclusion sur l'architecture conceptuelle

Les points clefs du style COMET sont :

- L'interconnection des trois graphes de COMET (LC, LM, PM) permet non seulement une séparation des préoccupations mais aussi un meilleur ancrage des BàO dans les méthodes de conception. On retrouve dans ces trois graphes, les niveaux d'abstraction classiques en IHM allant des tâches utilisateur aux mises en œuvre techniques. Les approches à base de modèles sont donc ici intégrées tout en ayant une approche et des objectifs BàO.
- La structure en graphe hiérarchique offre un moyen élégant de hiérarchiser les niveaux sémantiques des COMET par le niveau d'abstraction auquel ils sont ancrés (LC, LM ou PM). Par exemple, le niveau sémantique du LC de la COMET Entrelacement embarquée dans un PM de la COMET de messagerie instantanée est plus faible que le niveau sémantique du LC de la COMET de messagerie. Mais il est au même niveau sémantique que le LC de la COMET Activator elle aussi embarquée dans ce PM.
- Les PM assurent l'empreinte technologique des COMET : cette empreinte est extensible et non exclusive.
- La gestion des PM est dynamique : il est possible de brancher, débrancher, remplacer des PM à la volée ;
- Le style est applicable à toute préoccupation, qu'elle relève de la présentation ou du du niveau fonctionnel.

Le Tableau III-4 résume l'analyse conceptuelle de la BàO COMET.

Tableau III-4 : Analyse conceptuelle de la BàO COMET.

Niveau sémantique	Les COMET sont définies au niveau tâche. Elles sont affinées aux niveaux AUI et CUI/FUI. Le graphe de LC code la sémantique de l'application. Il est du niveau tâche. Le graphe des LM de présentation code l'AUI. Enfin les graphes des PM de présentation codent le niveau CUI/FUI.

Empreinte technologique	Une COMET peut avoir plusieurs PM dans des technologies différentes. Tous peuvent être actifs à un moment donné. Le graphe de COMET peut être rendu dans différentes technologies, de façon non exclusive et dynamique.
Extensibilité	De nouveaux PM peuvent être ajoutés, retirés ou remplacés à la volée. Certains peuvent être « sur mesure ».
Malléabilité	Un PM peut être substitué par un autre. Selon la technologie utilisée, des styles peuvent être appliqués (ex : CSS en HTML).
Contrôlabilité / Prévisibilité du rendu	Un ensemble de PM est fourni de base pour chaque COMET. Le rendu est prévisible. Il est aussi contrôlable car le concepteur peut à tout moment substituer un PM par un autre.

II.2. Architecture logicielle implémentationnelle

Cette section décrit l'architecture implémentationnelle de la BàO. J'ai choisi TCL comme langage de programmation. C'est un langage de script de haut niveau facilement interfaçable avec des librairies dynamiques C++. Il dispose de base d'une BàO d'interacteurs, TK, permettant de coder des interfaces WIMP. L'intégration des technologies présentées dans ce chapitre (AJAX, B207 et S207) s'est faite par l'utilisation soit simplement du langage TCL (génération de code AJAX) soit de librairies dynamiques générées automatiquement avec l'utilitaire SWIG (cas de B207 et S207)

Un atout des langages de script est la facilité à écrire des programmes générateurs de code et à interpréter ensuite ce code à la volée. Cette caractéristique a été particulièrement utile pour générer le corps de méthodes des API sémantiques. Un autre atout était la possibilité d'inspecter le code des procédures TCL.

L'implémentation des COMET se base sur une surcouche objet de TCL développée par François Bérard : gmlObject. gmlObject tire profit de TCL pour l'introspection de code.

L'architecture logicielle des COMET implique la définition, pour chaque COMET, des fonctions de son API sémantique. Ces fonctions assurent, en particulier, la propagation des événements entre facettes (LC→LM, LM→PM, etc.). Pour éviter une implémentation fastidieuse, un noyau fonctionnel commun est développé : CFC pour Common Functional Core. Le concepteur n'a alors qu'à appeler les procédures correspondantes pour assurer la propagation. L'API sémantique de la COMET est codée dans le CFC. Les accesseurs (« getter » et « setter ») à cette API sont générés pour le LC, les LM et les PM correspondant au CFC.

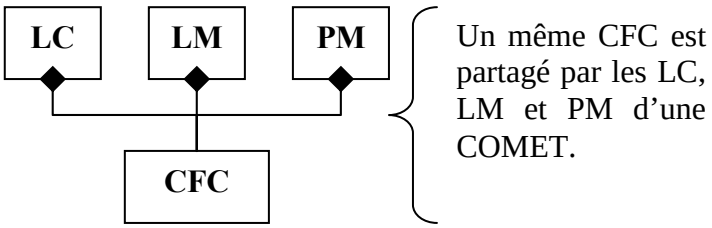


Figure III-47 : Fragment du modèle UML de l'architecture implémentationnelle des COMET sous gmlObject. Les éléments de COMET utilisent tous un même CFC (Common Functional Core).

Prenons l'exemple de la COMET Choice. Le CFC code l'API sémantique (« set_nb_max_choices », get_choices », etc.). Le code généré dans le LC, les LM et les PM fait appel au CFC. Les « getters » sont une indirection vers le CFC. Il en est de

même pour les « setters » qui, en plus, propagent ensuite l'appel aux éléments dont ils sont en charge (LC→LM, LM→PM). La génération de l'interface sémantique des LM de présentation est particulière puisqu'il faut en outre générer le code gérant le langage COMET/RE pour la multimodalité. Les seules méthodes qui ne seront pas entièrement générées automatiquement sont celles de l'API sémantique des PM. L'implémentation dépend des technologies ciblées. La génération ne peut pas être automatique. Typiquement, comment traiter « set_choices » dans un graphe de scène ?

II.2.A. AJAX pour le WEB

AJAX (Asynchronous JavaScript And XML) est un ensemble de technologies pour développer des applications WEB. AJAX combine XHTML, CSS, javascript (en particulier l'objet XMLHttpRequest) et DOM. Le terme « asynchrone » de AJAX provient du fait qu'une partie des traitements est désormais réalisée localement chez le client et non plus systématiquement du coté serveur. La connexion au serveur peut être réalisée par un script (utilisant XMLHttpRequest). La réponse est ensuite transformée (cas d'une réponse sous forme XML) et intégrée à la page via l'API DOM.

Du point de vue des COMET, l'implémentation de PM AJAX a nécessité de prendre en compte les particularités de cette technologie. Ainsi, fallait-il prendre en compte trois types d'entités : les clients WEB qui se connectent aux IHM produites par les PM AJAX, le graphe de COMET (en TCL) et le serveur WEB auquel se connectent les clients et qui fait le pont avec le graphe de COMET.

Les PM AJAX ont la particularité de ne manipuler aucun élément « concret » de XHTML. C'est-à-dire que, contrairement aux PM TK par exemple, les PM AJAX ne créent pas de boutons, de champs texte ou n'importe quel autre widget. Les PM AJAX se contentent de générer du code AJAX qui est ensuite transmis aux clients via le serveur. La difficulté était de transmettre au graphe de COMET le formulaire rempli par le client.

Prenons l'exemple d'un champ texte en XHTML. Celui-ci est modélisé par une balise « input » qui prend pour principaux attributs :

- type : la valeur est fixée à « text » pour un champ texte.
- value : le texte entré.
- name : le nom de la balise.

Une fois rempli par le client, le formulaire est transmis au serveur. Les données lui arrivent sous la forme de couples <identifiant de balise, valeur>. Il s'agit alors de trouver, dans le graphe de COMET, le PM concerné et la méthode à appeler sur ce PM. L'astuce que je propose tient au nommage des balises : les identifiants sont la concaténation du nom système du PM qui l'a généré et de la méthode à appeler sur ce PM pour le traitement des données. Par exemple, si la balise « input » est générée par un PM de COMET Texte du nom système « C_spec_PM_P_1 » alors l'identifiant de la balise (son nom ou son ID en XHTML, selon les balises) sera : « C_spec_PM_P_1__XXX__set_text ». La chaine de texte « __XXX__ » (arbitraire) sert de séparateur entre le nom de l'objet et le nom de la méthode.

L'unicité des identifiants HTML est assurée par la méthode de nommage. En effet :

- Les noms système des PM sont uniques.

- Un PM AJAX ne peut avoir qu'un père. Par déduction, il n'est donc rendu qu'une seule fois dans une page.

Grâce à cette méthode de nommage, les traitements sont correctement propagés aux PM. La Figure III-48 présente les différents messages échangés entre le client, le serveur et les COMET dans le cas d'un remplissage de formulaire.

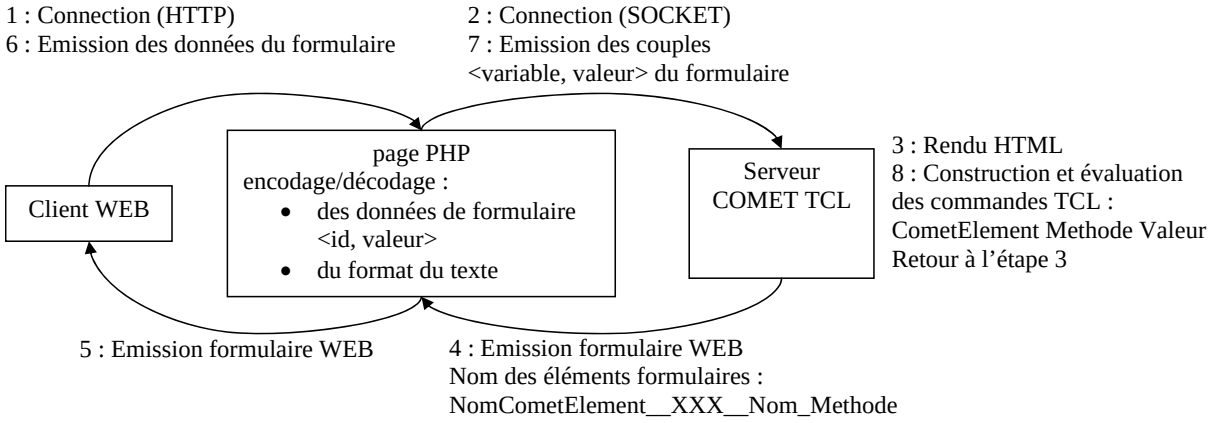


Figure III-48 : Vue générale de l'architecture des PM AJAX des COMET.

Examinons plus en détail le rendu des PM AJAX (Figure III-49).

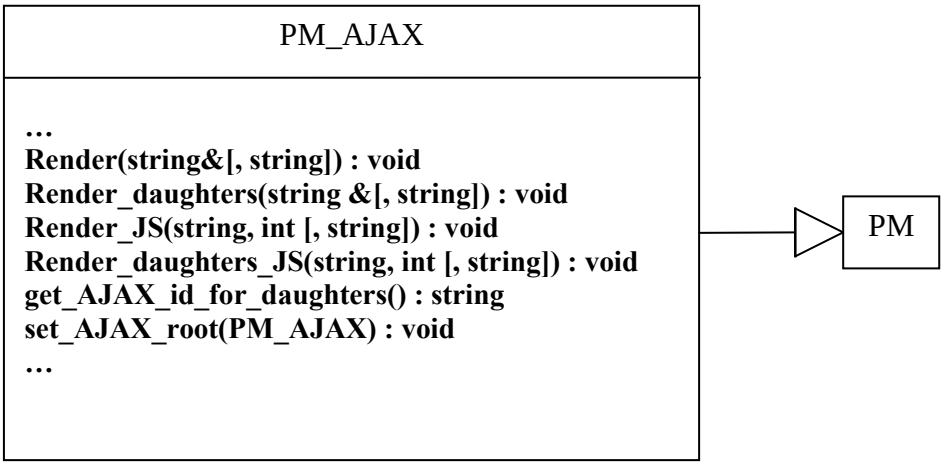


Figure III-49 : Une partie du modèle UML des PM AJAX.

Les méthodes « Render » et « Render_daughters » génèrent du code XHTML. Elles prennent en argument une référence sur une chaîne de caractère, à laquelle elles concatènent le code généré. Elles prennent optionnellement un deuxième argument, une chaîne de caractères d'espacements, pour gérer l'indentation du code généré. « Render_JS » et « Render_daughters_JS » se chargent de la génération du code javascript utile au PM (certains PM utilisent en effet javascript en plus de XHTML pour définir des présentations). L'entier passé en paramètre sert de marqueur pour éviter de générer plusieurs fois la même fonction. En effet, toutes les instances de PM d'une même classe utilisant javascript peuvent partager des fonctions javascript communes. Il est inutile de générer ces fonctions autant de fois qu'il y a d'instances de PM de ce type.

Par exemple, la méthode « Render » d'un PM d'activateur génère le code XHTML du bouton « **<input type="button" onclick="javascript:actionBouton(${PM}__XXX__prim_activate)" />** ». La fonction actionBouton permet de savoir sur quel bouton l'utilisateur a cliqué (les formulaires XHTML standards ne le permettent

pas). La méthode « Render_JS » du PM génère quant à elle le code de la fonction javascript actionBouton.

« set_AJAX_root » permet de fixer le nœud qui sera rendu en cas de requête AJAX (c'est-à-dire d'une requête initiée par un script javascript et non pas par l'utilisateur). « set_AJAX_id_for_daughter » permet de définir sous quelle balise ranger les informations renvoyées par une requête AJAX. Une requête AJAX a en effet la plupart du temps pour but de rafraîchir **une partie seulement** de la page WEB.

Un exemple d'utilisation d'une requête AJAX est fourni dans la COMET de CHAT. Le PM AJAX de cette COMET se charge de mettre à jour la page XHTML en fonction des changements dans l'historique du CHAT. Cela se réalise à l'aide d'une fonction javascript qui est générée dans la méthode « Render_JS » du PM AJAX de la COMET de CHAT. Cette fonction émet à intervalles réguliers (par exemple toutes les deux secondes) l'état de l'IHM (dernière modification de l'historique, message en cours). Le PM AJAX compare l'état reçu avec l'état courant. Il renvoie au client une mise à jour si besoin en fixant la balise sous laquelle placer la réponse (Figure III-50).

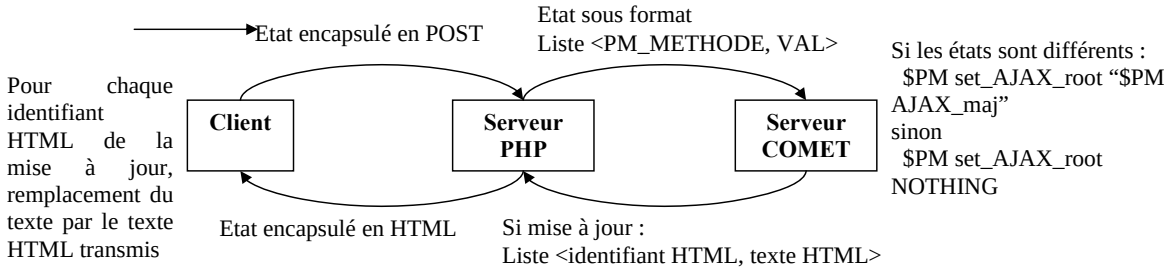


Figure III-50 : Maintien de la cohérence d'une page HTML en utilisant AJAX avec les COMET.

Un concepteur qui veut ajouter un PM AJAX doit implémenter les méthodes « Render » et éventuellement « Render_JS », si il y a nécessité de code javascript.

II.2.B. TK pour les interfaces classiques

TK est une BàO d'interacteurs WIMP fournie en standard avec TCL. La particularité de TK est que les noms des widgets codent aussi leur emplacement dans la hiérarchie de widgets TK. Par exemple, « .f » désigne un widget fils de la racine (« . »), .f.b désigne un widget fils de « .f ». Du point de vue des COMET, cela a de l'importance : un PM TK ne peut donc avoir qu'un seul père et les widgets TK ne peuvent être créés qu'au moment où le PM TK est branché à un père dont les widgets TK ont déjà été créés. Prenons par exemple le cas d'une COMET Activator ayant un PM TK, contenant un widget TK bouton. Il n'est pas possible de créer ce widget sans savoir où il sera branché puisque son nom détermine sa place dans la hiérarchie des widgets TK. Pour que le PM TK de la COMET Activator puisse créer ses widgets, il est nécessaire qu'il sache sous quel widget TK les brancher. L'API des PM TK est donnée en Figure III-51.

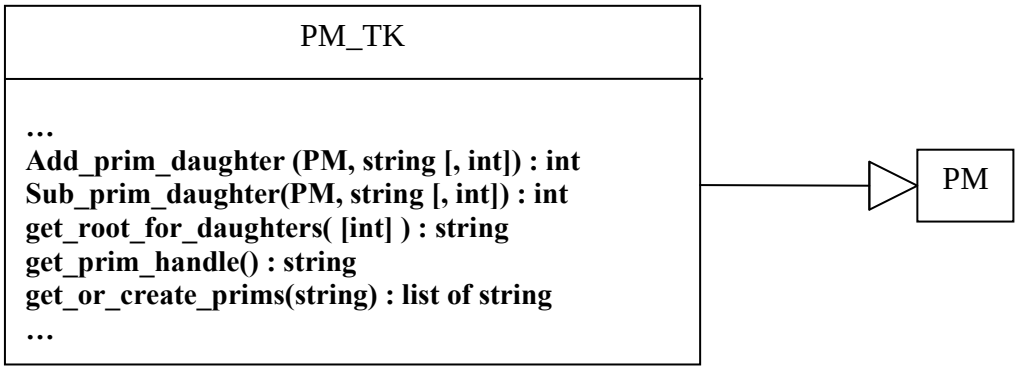


Figure III-51 : Une partie de l'API des PM TK.

L'API des PM TK utilise des méthodes spécifiques pour réaliser les branchements des widgets TK :

- « get_or_create_prims » prend en paramètre un chemin TK. Ce chemin correspond à un widget existant, qui sera le père des widgets TK du PM. Si ces widgets n'existent pas déjà, ils sont créés. Les widgets créés sont renvoyés sous la forme d'une liste.
- « Add_prim_daughter » permet d'ajouter aux widgets TK d'un PM les widgets TK passés en paramètre (argument string). Ces widgets TK correspondent à un PM fils (argument PM). Les widgets sont placés à un index éventuellement passé en paramètre (argument int). En pratique, cela revient à créer le ou les widgets TK. Dans le cas de l'Activateur, un appel du type « C_cont Add_prim_daughter (C_act, .C_cont_PM_P_f) » où C_cont est le nom d'un PM de Container auquel on veut brancher le PM d'un Activator C_act entraîne la création d'un bouton TK « .C_cont_PM_P_f.C_act_PM_P_bt ».
- « Sub_prim_daughter » se charge de débrancher les widgets TK. En pratique, cela revient à les détruire.
- « get_root_for_daughter » et « get_prim_handle » donnent accès respectivement aux widgets TK sous lesquels brancher les fils des PM, et aux widgets TK « racines » d'un PM donné.

Du fait de la convention de nommage des widgets TK, il est impossible qu'un widget soit partagé par plusieurs autres widgets. Tout comme il est difficile de déplacer des parties d'interfaces dans la hiérarchie TK (par exemple pour passer le contenu d'un canevas dans une fenêtre). L'utilisation des PM TK permet de surmonter ces limitations. En effet, il est possible de créer plusieurs PM TK pour une même COMET. Cela permet de surmonter la limitation du partage d'un widget par la création de plusieurs widgets similaires, dont la cohérence est assurée par l'architecture logicielle des COMET. De la même façon, la limitation quant au déplacement d'une partie de l'interface dans la hiérarchie TK est surmontée par la surcharge des opérateurs de branchement et débranchement de PM. Ceux-ci prennent en compte la destruction et la création des widgets TK concernés par le déplacement.

La Figure III-52 illustre comment une vue multiple peut être réalisée à l'aide des COMET. Quatre containers partagent une même COMET de choix d'un mois. Puisqu'un PM TK ne peut avoir qu'un seul père, quatre PM de présentation TK sont instanciés pour la COMET de choix d'un mois. A l'aide du style CSS++ (décrit dans la section suivante), il est ensuite possible de contrôler la présentation de chacun d'eux.

a) b)

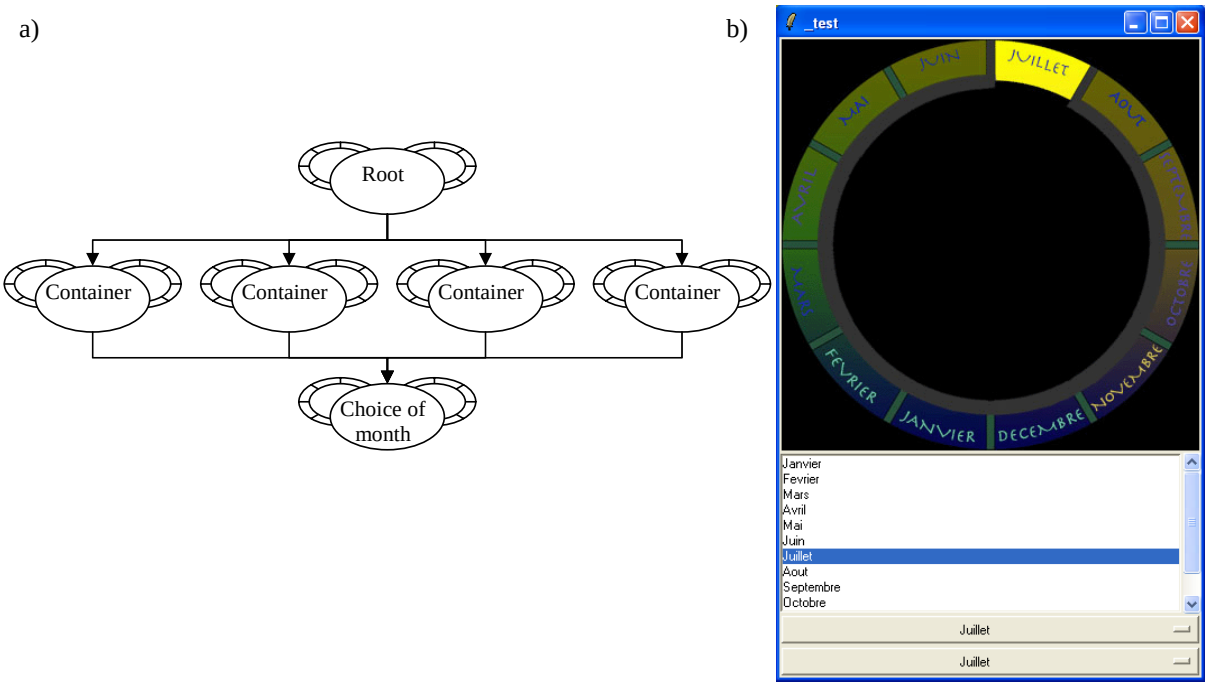


Figure III-52 : a) Graphe de COMET. Le Choix du mois est partagé entre 4 containers. b) Représentations TK. Une représentation différente a été choisie pour les 3 premiers PM, la quatrième est similaire à la troisième. La cohérence entre les PM est assurée par le LM de la COMET de choix.

Les PM TK montrent qu'il est possible de dépasser certaines limitations des BàO WIMP grâce à l'architecture COMET. Un concepteur qui veut ajouter un PM TK doit surcharger, outre les méthodes de l'API sémantique, la méthode « get_or_create_prims ».

II.2.C. B207 pour les interfaces post-WIMP

B207 est une BàO post WIMP expérimentale que j'ai développée. J'ai choisi d'illustrer la possibilité de concevoir des IHM post WIMP avec les COMET à l'aide de la BàO B207. Ceci montre l'empreinte technologique large des COMET et leur capacité à couvrir le post-WIMP.

B207 permet de manipuler des IHM 2D à l'aide de plusieurs pointeurs. B207 décrit l'IHM à l'aide d'un graphe de scène. Chaque nœud contient une transformation et/ou des commandes d'affichage. A la manière d'UBIT, B207 gère un contexte généralisé qui est propagé le long des nœuds du graphe de scène. Un graphe de scène B207 est en général un DAG mais peut parfois contenir des boucles pour peu qu'un des nœuds de la boucle gère une condition d'arrêt. La capture d'écran de la Figure III-53 montre une IHM réalisée en B207. En bas à droite se trouve une fenêtre radar (Overview), implémentée à l'aide d'une vue multiple (la fenêtre prend pour fils l'IHM affichée au centre de l'écran). Au centre, se trouve une « loupe » (Fiche de zoom ovale) dont la « lentille » peut être réglée avec l'IHM à gauche de l'écran (la courbe). La loupe crée une boucle dans le graphe de scène puisque son fils est aussi son père. Sous la « loupe », se trouvent des fenêtres du démonstrateur CamNote++ (télécommande et visualisateur décrits dans le chapitre suivant) ainsi qu'une CLI (Command Line Interface) de l'interpréteur TCL.

B207 propose des fonctions d'animation dans le temps. Il est ainsi possible de réaliser des fondus, des déplacements ou des zoom.

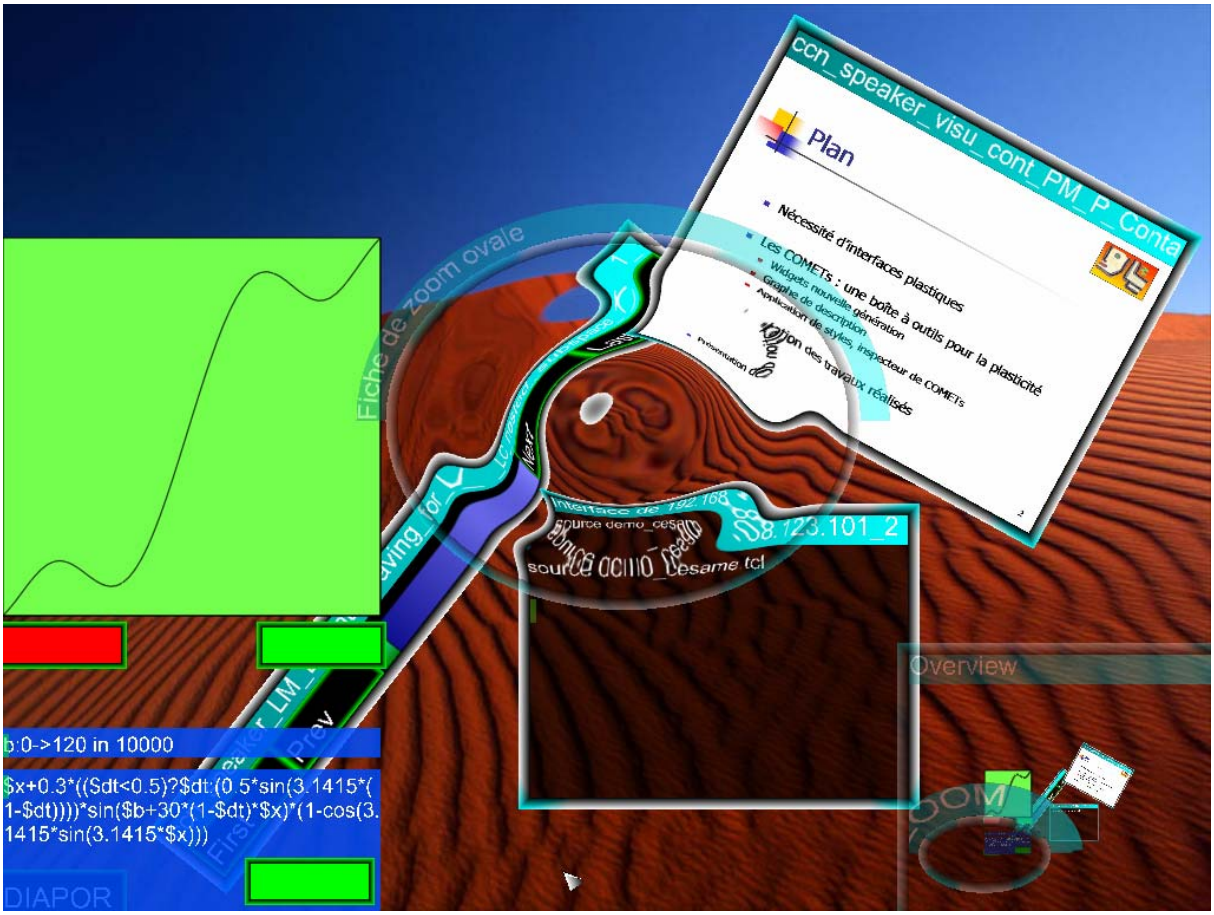


Figure III-53 : Exemple d'une IHM en B207.

Puisque les nœuds d'une interface B207 peuvent être partagés entre plusieurs nœuds père, il est possible, du point de vue des COMET, de créer des nœuds B207 dès la création des PM B207. Il résulte aussi de cela qu'un même PM B207 peut avoir plusieurs pères.

L'intégration de B207 dans les PM de COMET n'a pas posé de problème particulier. L'API des PM B207 est la même que celle des PM TK. En particulier, toutes les méthodes qui concernent la gestion des primitives (B207 au lieu de TK) se retrouvent dans l'API des PM B207. Seule leur implémentation diffère. Un concepteur qui voudrait créer des PM B207 devrait simplement surcharger, outre l'API sémantique, le constructeur pour créer les nœuds B207 et les enregistrer comme primitives.

II.2.D. S207 pour les interfaces vocales

S207 est une BàO d'interacteurs vocaux développée pendant ma thèse. Contrairement à B207, S207 a été spécifiquement conçue pour être intégrée dans l'architecture COMET. L'objet de S207 est de montrer que les COMET ne sont pas limitées au graphique.

S207 permet de reconnaître des commandes sur le graphe de COMET. Les commandes sont analysées sous la forme d'une chaîne de texte. Idéalement, cette chaîne de texte devrait être extraite à partir d'un enregistrement vocal. Cependant, le manque de fiabilité des programmes de reconnaissance vocale et la complexité de mise en œuvre de ces solutions m'ont amené à ne prendre concrètement en compte que des phrases saisies dans un champ texte. Cette limitation ne remet pas en cause l'approche adoptée dans S207. Elle est seulement limitative à l'usage. Il est, en effet, moins commode de saisir du texte au clavier que de l'énoncer vocalement.

En sortie de S207, j'ai utilisé la [76 Speech API] de Microsoft pour générer des réponses vocales. Cette technologie est plus mûre que la reconnaissance vocale et donne des résultats satisfaisants tout en restant simple d'utilisation.

Une interface S207 est un graphe de nœuds (les PM). Dans ce graphe, je distingue la racine. C'est au niveau de la racine qu'arrivent les commandes à évaluer. C'est la racine qui décide des traitements à opérer une fois la commande évaluée. La racine évalue la commande en s'appuyant sur le graphe des nœuds PM S207. Pour cela, chaque PM S207 implémente la méthode *Analyse* qui prend en paramètre une chaîne de texte et renvoie une liste d'interprétations possibles. Une interprétation est une liste de couples <PM S207, mots reconnus par ce PM>. La Figure III-54 présente la méthode d'analyse de base qui est fournie dans S207. Un PM essaie de reconnaître une expression dans la phrase fournie en paramètre (*str_name*). S'il y arrive, il essaie de faire reconnaître le reste de la phrase à ses descendants qui renvoient une liste des interprétations qu'ils ont trouvées. La phrase d'origine est aussi passée intégralement aux descendants du PM. En effet, l'expression reconnue par le PM peut aussi être reconnue par ses descendants et engendrer une autre interprétation. Toutes les interprétations trouvées sont renvoyées en résultat.

Pour reconnaître des éléments de phrase, les PM S207 se basent en partie sur le vocabulaire fourni par la COMET elle-même. En particulier, son nom (nom à usage de l'utilisateur- ce n'est pas le nom système) et ses classes peuvent être utilisées. Par exemple, dans le cas d'un PM d'activateur de nom « Diaporama », la reconnaissance pourra se faire avec l'expression régulière « *(action|activate|trigger)? *(Diaporama) *(.*)$ » (selon la syntaxe des expressions régulières en TCL).

Une fois la phrase analysée, une liste d'interprétations est retournée. C'est à la racine de déterminer quelles sont, parmi ces interprétations, les meilleures. La décision peut être faite avec l'utilisateur. L'implémentation actuelle est simple. Elle choisit les interprétations qui font appel à un maximum de PM.

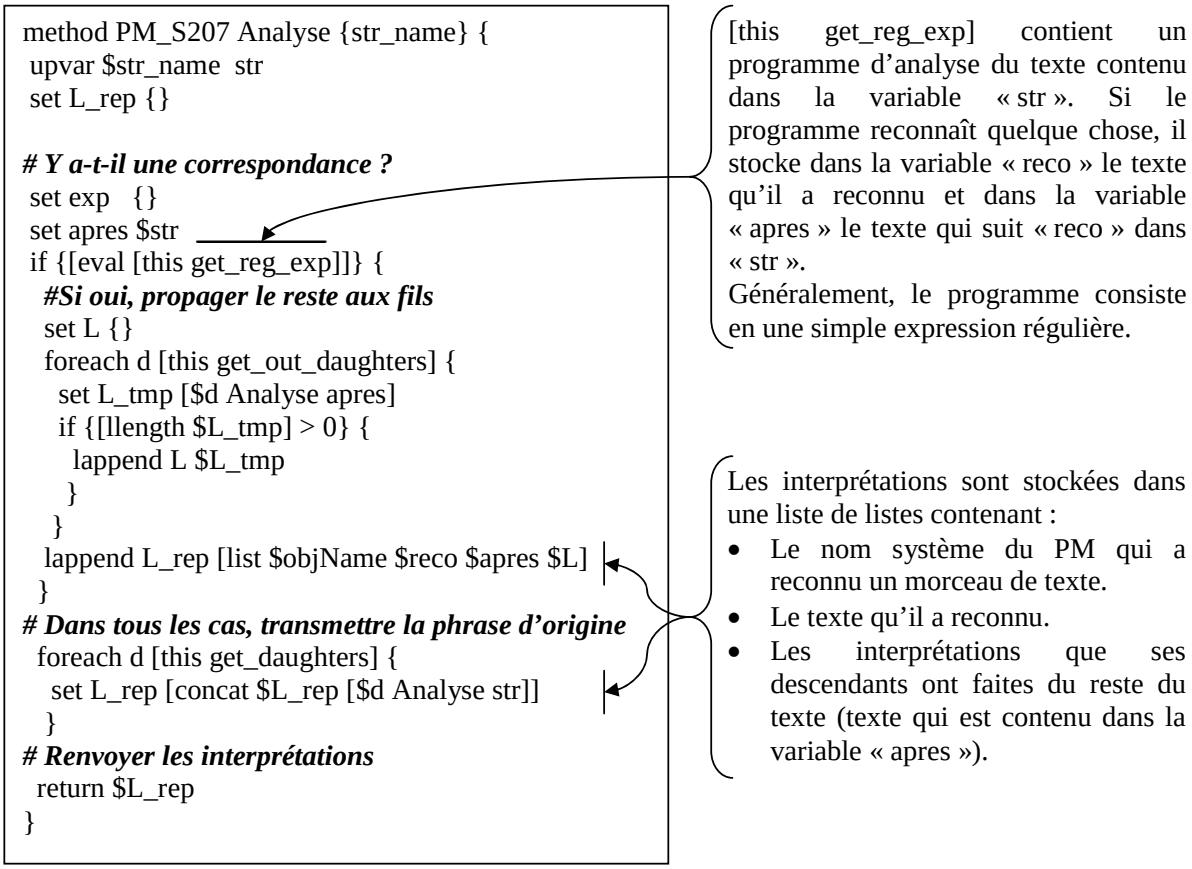


Figure III-54 : La méthode *Analyse* des PM S207.

Une fois la ou les interprétations déterminées, il faut les mettre en œuvre. Pour cela, la racine renvoie à chaque nœud la partie de la phrase qu'il a reconnue (méthode « Process_TXT (string) »). Les nœuds évaluent ces parties de phrase et les traitent. Par exemple, le PM S207 d'activateur qui reçoit la partie de phrase « activate … » doit déclencher sa méthode « activate ». Si la phrase contient une question, le PM y répond en renvoyant un texte contenant la réponse (ex : « what time is it ? » → « It is 11 P.M »). Dans le cas d'une action (ex : « activate »), il n'y a pas de réponse à donner. La politique de la racine est alors de répéter ce qu'elle a compris de la phrase énoncée puis de l'appliquer (ex : « well…let's activate diaporama please » → « activate diaporama »).

Le concepteur qui veut ajouter un nouveau PM S207 n'a que deux méthodes à surcharger : « Analyse » et « Process_TXT ».

S207 montre qu'il est possible d'implémenter des interfaces vocales simplement avec les COMET. La grammaire est implicitement déclarée avec la structure de graphe de l'application. L'ajout ou le retrait de COMET est immédiatement répercuté sur la reconnaissance des phrases puisque celle-ci est entièrement basée sur le graphe de PM S207.

II.2.E. Conclusion sur la partie implémentationnelle

Les différentes technologies utilisées dans les PM (TK, AJAX, B207 et S207) montrent que le modèle d'architecture logicielle employé dans les COMET autorise l'emploi de technologies variées. La force de l'approche COMET est de permettre l'utilisation simultanée de ces technologies. Ainsi, il est facile de combiner une présentation graphique (ex : TK) avec une présentation vocale (ex : S_207). L'architecture COMET

permet en outre de dépasser certaines limitations imposées par des technologies. Par exemple, TK ne permet pas de réaliser des vues multiples ou de facilement déplacer des parties d'interfaces dans la hiérarchie de widgets. L'utilisation des PM TK palie cette limitation.

III. GDD : Un réseau sémantique comme annuaire

Cette contribution a donné lieu aux publications suivantes : [34 Demeure 2002], [35-36 Demeure 2003], [8 Balme 2004], [18 Calvary 2004], [41 Demeure 2006].

Le GDD (Graphe Des Descriptions) joue le rôle d'annuaire de services pour les systèmes interactifs. Le GDD n'est pas limité aux COMET. Il permet de classer aussi bien des systèmes interactifs à base de COMET que n'importe quel autre système interactif. Le GDD est un réseau sémantique ([94 SOWA 1976], [95 SOWA 2000]). Son but est de classer des descriptions de systèmes interactifs. Par description de système interactif, j'entends n'importe quel modèle de système interactif. La description peut être formelle et/ou informelle, complète ou non. Elle peut correspondre à un système existant ou pas

[95 SOWA 2000] définit un réseau sémantique comme étant une notation graphique permettant de représenter des connaissances sous la forme de nœuds et d'arcs interconnectés. Les premières utilisations des réseaux sémantiques en informatique proviennent du domaine de l'Intelligence Artificielle et de la Traduction Automatique. Toutefois, le concept de réseau sémantique est depuis longtemps utilisé en philosophie, psychologie et linguistique ([95 SOWA 2000]). Le premier exemple connu de réseau sémantique provient du philosophe grec Porphyry au troisième siècle. Ce réseau avait pour but d'illustrer la méthode Aristotélicienne de catégorisation : spécifier un type général (*genus)* puis expliciter les différences (*differentiae*) qui lient ce type à ses sous-types.

SOWA identifie cinq types de réseaux sémantiques :

- Les réseaux de définitions (Definitional networks) mettent l'accent sur les relations « sorte de » entre concepts. Ces réseaux, aussi appelés hiérarchies de généralisation (generalization hierarchy) sont adaptés à la représentation de l'héritage de propriétés. Les sous-types héritent des propriétés de leurs super-types.
- Les réseaux d'assertions (Assertional networks) sont conçus pour exprimer des propositions. Par exemple : « Tous les fermiers qui ont des ânes les battent. ».
- Les réseaux d'implications (Implicational networks) sont un type de réseau d'assertions qui utilisent principalement l'implication comme relation entre les nœuds. Ces réseaux sont utilisés pour faire des déductions. Les réseaux bayésiens sont une illustration des réseaux d'implications.
- Les réseaux exécutables (Executable networks) incluent des mécanismes qui peuvent entraîner la modification du réseau lui-même. Les réseaux exécutables peuvent eux-mêmes se modifier, contrairement aux réseaux « statiques » qui ne peuvent être modifiés que de l'extérieur. Trois types de mécanismes sont généralement utilisés pour cela : le passage de messages entre les nœuds, les procédures attachées aux nœuds et les transformations de graphes.

- Les réseaux apprenants (Learning networks) peuvent construire, étendre et pondérer leur structure à l'aide d'exemples.

Le GDD est un réseau sémantique de définitions (section 1). Il joue le rôle d'annuaire de services pour les COMET (section 2).

III.1. Le GDD, un réseau sémantique de définitions

Au sens de SOWA, le GDD est un réseau sémantique de définitions. Les noeuds représentent des descriptions de systèmes interactifs. Les descriptions sont basées sur le cadre de référence CAMELEON. Les IHM sont décrites en termes de concepts du domaine, tâches utilisateur, interfaces abstraites, concrètes et finales. Les nœuds sont typés du niveau d'abstraction de la description. Par exemple, une IHM définie par un modèle de tâche sera typée tâche. Si la description se fait à plusieurs niveaux d'abstraction, le type du nœud est le niveau le plus concret. Par exemple, si une description est faite aux niveaux tâche, AUI et CUI, la description sera typée « CUI ».

Les noeuds du réseau sont interconnectés par des relations. Ces relations expriment les propriétés des transformations qui unissent ces nœuds. Par exemple, la relation de spécialisation entre un nœud A et un nœud B (B spécialise A) exprime la propriété que B peut être utilisé partout où A peut l'être. Les transformations sont un point clé pour la réutilisation des connaissances acquises lors de la conception des IHM ([86 Puerta 1999]). Le fait de poser des propriétés sur ces transformations permettra de naviguer utilement dans le GDD.

J'identifie neuf propriétés de transformations [41 Demeure 2006] :

- Inheritance : y hérite de x si y affine x. Cette relation peut être totale ou partielle, exclusive ou non. Totale signifie qu'il n'existe aucun z autre que les y déjà définis tels que z hérite de x Si la relation n'est pas totale, elle est dite partielle. Exclusive signifie que si y et z héritent de x alors il n'existe pas de w tel que w hérite (même indirectement) de y et de z. Elle est dite non exclusive sinon.
- Restriction : La restriction est un cas particulier d'héritage. Elle signifie que y est un sous cas de x. En d'autres termes, y peut être vu comme un x dans certains cas seulement.
- Specialization : La spécialisation est un autre cas particulier d'héritage. Elle préserve la propriété de substitution Si y spécialise x alors y peut être utilisé partout où x l'est.
- Extension : L'extension est une spécialisation avec ajout de capacités. Si y étend x, alors y peut être utilisé partout où x l'est mais y est enrichi de nouvelles descriptions. Notons que la relation d'extension est toujours partielle.
- Concretisation : La concrétisation est une forme particulière de spécialisation. Dire que y concrétise x signifie que y est fidèle à la description de x mais l'enrichit de descriptions d'un niveau plus concret. Par exemple, si x est décrit au niveau tâche, un y concrétisera x, s'il correspond à cette tâche et est au moins décrit au niveau AUI. La concrétisation est aussi dite « réification » dans le cadre de référence CAMELEON.
- Implementation : L'implémentation est une concrétisation particulière. L'enrichissement se fait par l'ajout d'une description du système qui soit

exécutable ou interprétable, c'est-à-dire par l'ajout du code. Par exemple, si y implémente x et que x modélise la tâche calculer, alors y peut être une calculette fonctionnant sous environnement JAVA.

- Agrégation : L'agrégation est une transformation autre que l'héritage. y est aggrégé à x signifie que y peut être vu comme un sous système de x. Dans ce cas, les correspondances entre x et y sont décrites dans la relation d'agrégation. En particulier on décrit où et comment y prend place dans x.
- Composition : La composition est un type d'agrégation particulier. Elle exprime une dépendance fonctionnelle entre le y encapsulé et le x encapsulateur : si x est déruit, y l'est également.
- Use : L'utilisation est un type de composition particulier qui exprime l'inverse de l'encapsulation. Si x utilise y, la survie de de y n'est pas conditionnée par celle de x.

La description des transformations montre qu'elles ne sont pas indépendantes. La Figure III-55 les organise dans un graphe d'héritage.

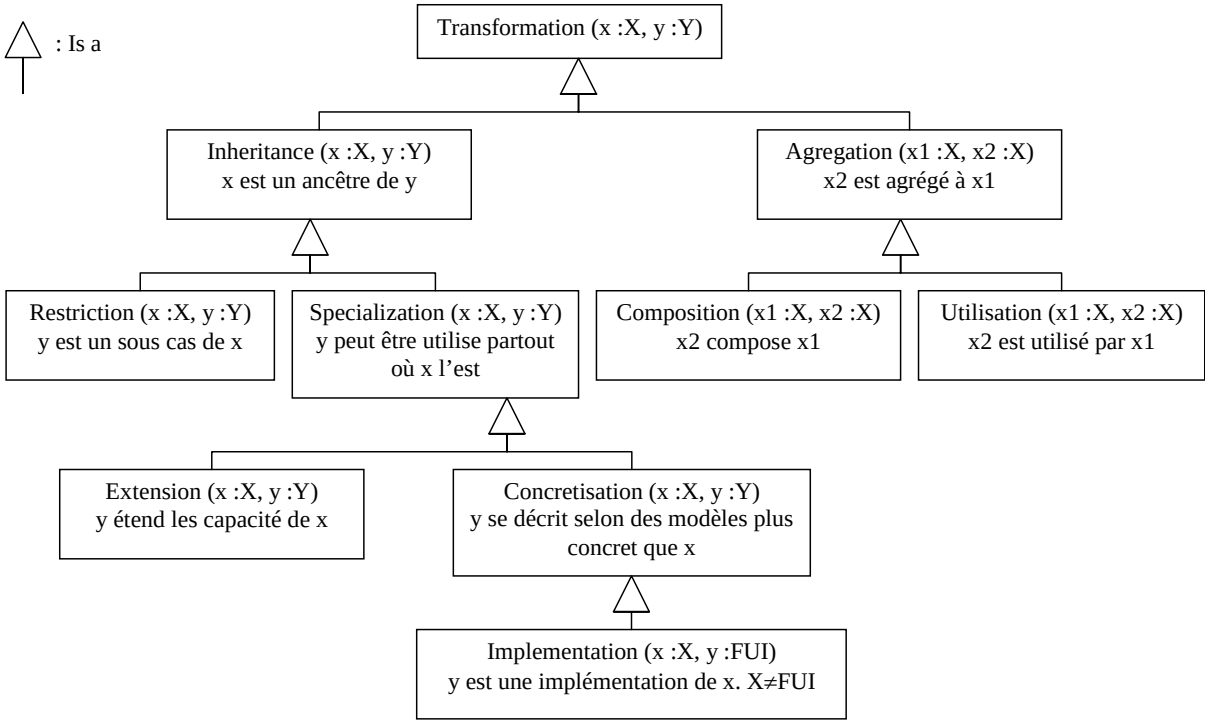


Figure III-55 : Graphe d'héritage situant les relations du GDD entre elles.

Compte tenu de la taille importante du GDD, je ne le présente pas en entier. Je présente deux sous parties : une liée à la tâche « Choisir » et l'autre aux COMET dans le GDD.

III.1.A. « Choisir » dans le GDD

L'exemple du Choix est intéressant à plusieurs titres : il est simple mais néanmoins riche ; il est utilisé dans presque tous les systèmes interactifs ; de nombreux interacteurs l'implémentent (menu déroulant, liste déroulante, boutons radios, cases à cocher, accumulateurs, menus en fleur, mais aussi des interacteurs spécifiques à des concepts particuliers tels que le calendrier, le choix de couleurs, le choix de fichiers, etc.).

Dans cette variété de choix, le concepteur est traditionnellement guidé par les critères d'utilisabilité. Par exemple, les nombres d'éléments à choisir ou d'éléments disponibles

peuvent intervenir, mais aussi la variabilité des choix ou encore la superficie requise en surface d'affichage [79 Nogier 2005]. Des recommandations ergonomiques existent mais elles ne sont pas embarquées dans les environnements de développement. A noter qu'à l'époque des générateurs d'interfaces, de nombreux travaux ont porté sur l'informatisation de ces recommandations ergonomiques [101 Vanderdonckt 1993] [43 Farenc 1997]. Il s'agit, par le GDD, de décrire finement tout élément d'IHM de façon à ce que les règles soient applicables.

La Figure III-56 montre comment les descriptions de systèmes de type « Choisir » sont organisées au sein du GDD. Pour des raisons de lisibilité, les nœuds d'un même niveau d'abstraction sont regroupés au sein d'une même zone colorée. Chacune de ces zones est marquée d'un libellé indiquant le niveau d'abstraction dont elle relève : C&T pour Concepts et Tâche, AUI pour Interface Abstraite, CUI pour Interface Concrète et FUI pour l'interface finale.

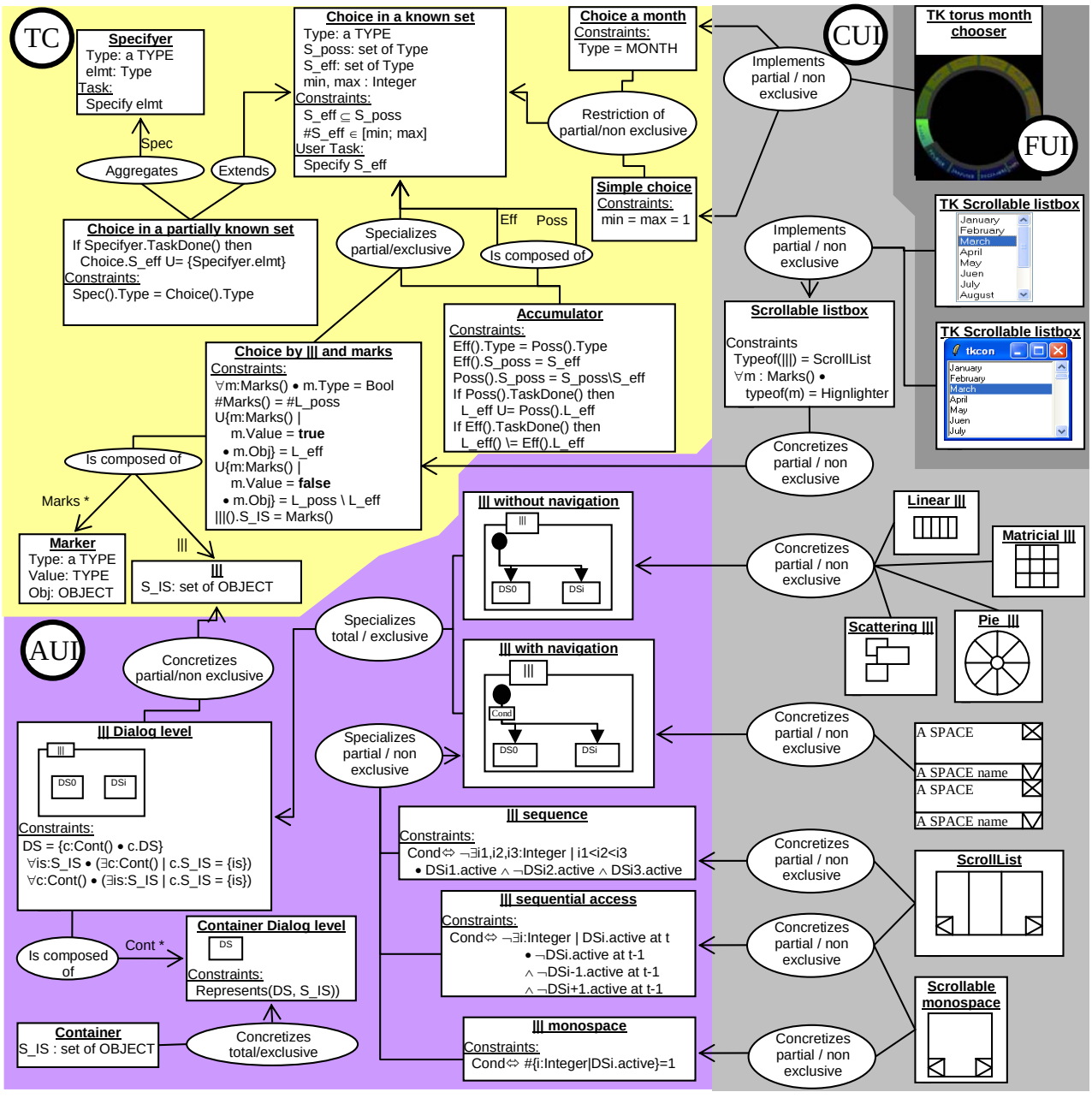


Figure III-56 : Une partie du GDD pour la tâche « Choisir ». Extrait de [41 Demeure 2006].

Une description informelle est donnée pour chaque noeud. Par exemple, au niveau C&T, pour la tâche « Choice in a known set » (choix dans un ensemble connu d'éléments), la description précise que :

- Les éléments à choisir sont tous d'un certain type « Type ».
- Les éléments qu'il est possible de choisir sont regroupés dans l'ensemble « S_poss » (S pour Set et poss pour possibles).
- Les éléments choisis sont regroupés dans l'ensemble « S_eff » (S pour Set et eff pour effectifs).
- Deux entiers « min » et « max » désignent le nombre minimal et maximal de choix effectifs.
- Des contraintes existent : (1) S_eff doit être inclus dans S_poss (2) le cardinal de S_eff est compris entre min et max.

La tâche « choix d'un mois » est une restriction de la tâche choix au sens où le type des éléments à choisir est contraint (contrainte : Type = MONTH). Une interface finale TK est disponible pour ce choix d'un mois (TK Torus month chooser). Cette FUI est une implémentation non seulement du « choix d'un mois » mais aussi du « choix simple ». Le « choix simple » (Simple Choice) est une restriction du choix telle qu'un seul élément peut être choisi (contrainte : min=max=1).

Le système « Choice in a known set » peut être spécialisé (partiellement et exclusivement) de deux façons : en accumulateur et en choix par entrelacement/marqueurs. Ces deux spécialisations sont des façons de réécrire le problème du choix au niveau C&T. L'accumulateur s'appuie sur deux sous choix : le premier gère les éléments non choisis, l'autre les éléments choisis.

Les accumulateurs séparent les éléments choisis des éléments non choisis. Récursivement, les accumulateurs encapsulent deux systèmes « Choisir » : l'un porte sur les éléments non choisis qui passeront en éléments choisis (Poss sur la Figure III-56) ; l'autre porte sur les éléments choisis qui passeront en éléments non choisis (Eff sur la Figure III-56). Typiquement, un accumulateur est implémenté au niveau final à l'aide de deux listes à choix multiples.

Les choix par entrelacement et marqueurs encapsulent deux types de systèmes : un entrelacement et des marqueurs. Un entrelacement permet de présenter un ensemble de systèmes à l'utilisateur. Celui-ci peut les réaliser dans n'importe quel ordre et passer de l'un à l'autre à n'importe quel moment. Les exemples typiques d'entrelacement sont l'entrelacement en colonne, ligne, matrice, les onglets, les menus ou encore la barre des tâches sous windows. Un marqueur est fondamentalement un booléen associé à un objet (par exemple un autre système). Dans le cas du système « Choisir », les marqueurs sont associés aux éléments à choisir. La marque est vraie si l'élément est choisi, faux sinon. Les exemples typiques de marqueurs sont les boutons radios, les cases à cocher, les rectangles englobant ou encore les mises en surbrillance. Le système de choix par entrelacement et marqueurs entrelace des marqueurs, chacun étant associé à un et un seul élément à choisir.

Le système « Entrelacement » (|||) permet d'illustrer la relation de concrétisation. Au niveau AUI, l'entrelacement est concrétisé en un espace de dialogue contenant les sous-espaces. Chaque sous-espace est associé à un et un seul élément à entrelacer (le modèle d'AUI de cet élément). Les relations entre la modélisation C&T et la modélisation AUI sont données dans le nœud décrivant l'AUI (||| Dialog level).

Une spécialisation (totale et exclusive) de l'entrelacement au niveau AUI (||| Dialog level) divise les entrelacements en deux catégories : ceux qui obligent à rendre tous les éléments à choisir et ceux qui permettent de n'en rendre qu'une partie, le reste étant accessible par navigation. Les entrelacements par navigation sont eux-mêmes divisés selon des propriétés qu'on peut exprimer sur le type de navigation qu'ils offrent. Par exemple, certains obligent à accéder de façon séquentielle aux éléments entrelacés. D'autres ne peuvent rendre qu'un élément à la fois, etc. Les entrelacements sans navigation peuvent être distingués au niveau CUI selon des critères géométriques. Les éléments peuvent être placés en ligne, en matrice, en rond ou encore de façon éparpillée.

L'exemple du choix par entrelacement montre que le GDD peut non seulement être utilisé comme outil de classification des systèmes interactifs mais aussi comme outil pour l'exploration de nouveaux systèmes. En effet, ce système de choix est composé par un entrelacement et des marqueurs. En faisant le produit cartésien de toutes les CUI d'entrelacement et de toutes les CUI de marqueurs (à la ACE), on peut explorer des possibilités qui n'avaient encore jamais étés mises en œuvre jusqu'ici. Par exemple, un entrelacement avec navigation séquentielle combiné avec des marqueurs de type cases à cocher, ou bien un accumulateur composé d'une liste d'éléments à choisir et d'un menu en fleur des éléments choisis.

La section suivante montre que les COMET peuvent être classées dans le GDD.

III.1.B. Les COMET dans le GDD

Les COMET peuvent être placées dans le GDD. Il s'agit d'y ventiler les descriptions des LC, LM et PM. Je ne donne pas de métamodèles pour ces éléments. Je montre juste le principe. La Figure III-57 montre comment les descriptions des LC, LM et PM sont organisées au sein des quatre niveaux d'abstraction. Les descriptions de LC prennent place aux niveaux C&T et FUI (LC implémenté dans un certain langage). Les descriptions de LM prennent place aux niveaux C&T (LC composé de LM), AUI (pour les LM de présentation) et FUI (LM implémenté). Les descriptions de PM prennent place aux quatre niveaux : LC (LM composé de PM), AUI et CUI (pour les PM de présentation) et enfin FUI (PM implémenté avec une certaine BàO).

L'illustration est faite sur la COMET Choice. Pour des raisons de lisibilité, il n'est pas possible de présenter l'intégralité du réseau sémantique qui décrit le choix (Choice). Notons que les descriptions de COMET peuvent être reliées à des descriptions autres que des descriptions de COMET. Par exemple, « LC Choice » est relié à « Choice in a known set » (cf Figure III-56) comme une spécialisation de celui-ci. En pratique, cela signifie que « LC Choice » hérite de l'API de « Choice in a known set ». C'est l'API sémantique du LC. « LC Choice » hérite aussi, par restriction, de l'API de LC pour se conformer au modèle d'architecture COMET. La restriction entre « LC Choice » et LC porte sur le type de LM qu'accepte « LC Choice » : seul un « LM Choice » peut être branché.

Un des PM possibles pour le choix est le « PM Choice Accumulator ». La Figure III-56 montre que ce PM est composé de deux LC Choice : un pour les choix faits (Eff), l'autre pour le reste (Poss). La décomposition est bien du niveau C&T et correspond à l'architecture conceptuelle décrite dans la Figure III-27.

Les niveaux AUI et CUI sont illustrés avec des modèles de PM Marker. Au niveau CUI, plusieurs types de PM Marker sont décrits : boutons radios, cases à cocher, soulignage, encadrement. Des implémentations de ces PM Marker se retrouvent au niveau FUI. Les

implémentations utilisent TCL et différentes technologies de rendus (HTML, TK, B207).

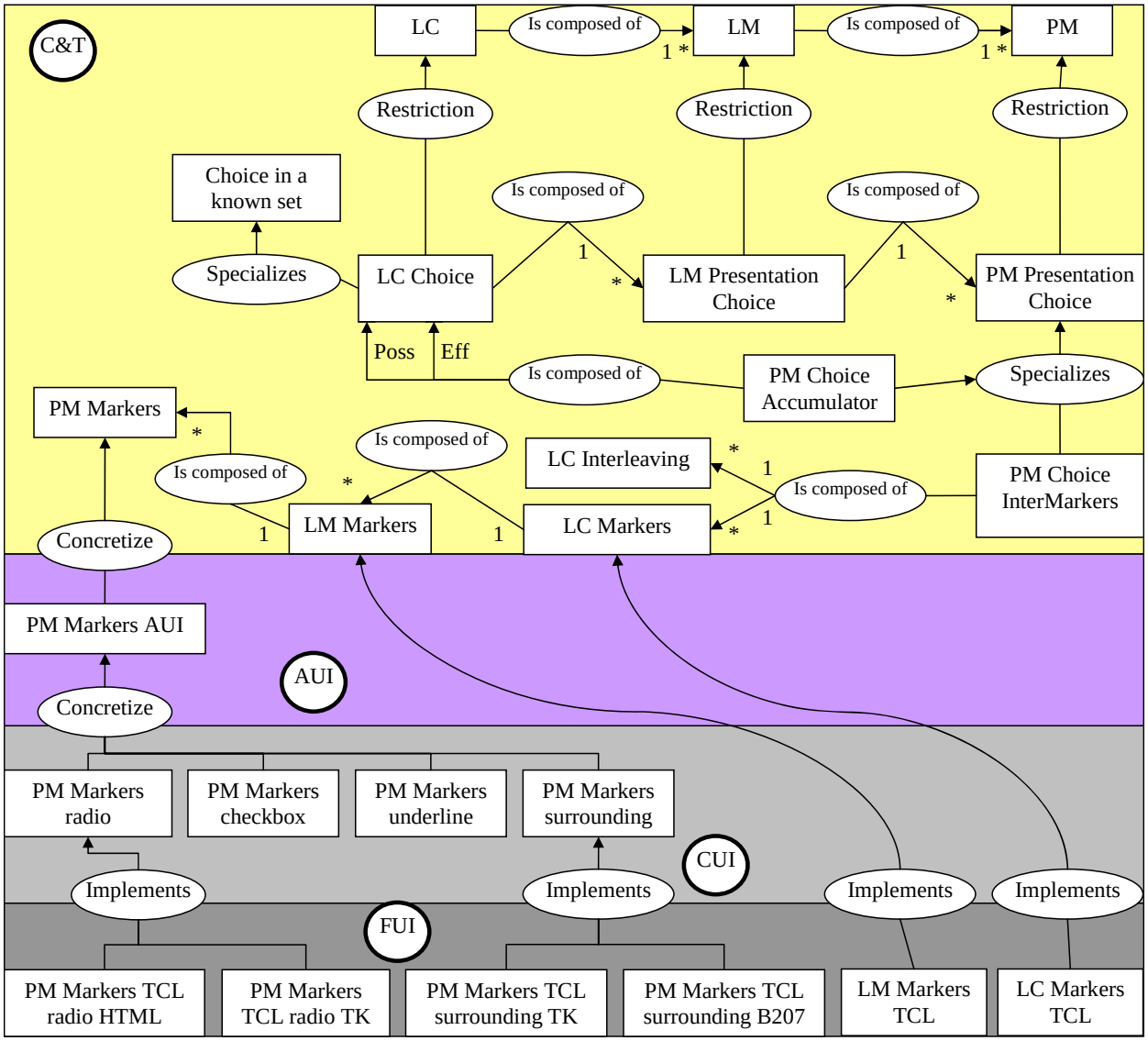


Figure III-57 : Organisation des COMET dans le GDD.

III.2. Le GDD, un annuaire de systèmes interactifs

Le GDD peut être exploité comme annuaire de systèmes interactifs. A la manière des annuaires de services, il est possible de spécifier des requêtes pour retrouver des systèmes particuliers. L'avantage du GDD par rapport à des systèmes comme UDDI ou l'annuaire d'Amigo est que les relations entre systèmes sont déjà établies. Le désavantage est bien entendu que cela implique une classification manuelle ou semi-automatique des systèmes interactifs. Cet inconvénient est d'autant plus important que le nombre de systèmes est grand. L'autre inconvénient est l'absense de certifcation du GDD : la classification est-elle correcte ? Rien ne le prouve. Les avantages toutefois sont importants :

- On évite les erreurs des classements automatiques pointées dans l'état de l'art. On considère le GDD comme étant l'oracle au problème du classement des systèmes interactifs entre eux.

- On n'impose pas de langage de description particulier. Tout peut être utilisé (UML, WSDL, CTT, statecharts, réseaux de pétri, etc.). Ce dernier point est particulièrement important pour les niveaux de description AUI, CUI et FUI puisqu'aucun standard n'existe à ce jour, contrairement à UML pour la modélisation des concepts ou, dans une moindre mesure, CTT pour les tâches.
- Le GDD permet de répondre à des requêtes comme :
  - Donne moi les systèmes de type « Choisir » qui assurent un rendu sans navigation de tous les éléments à choisir.
  - Donne moi l'équivalent pour la technologie JAVA de ce système interactif en technologie AJAX.
  - S'il n'existe pas d'implémentation AJAX d'un système donné, donne moi les descriptions les plus précises de ce système (de sorte à pouvoir ensuite générer une version AJAX).

Pour exploiter le GDD, je propose un langage de requêtes nommé GDD_Q (Q pour Queries). GDD_Q permet de retrouver des nœuds du GDD en partant d'une racine et en exprimant un chemin en termes de nœuds et de relations à franchir. Le Tableau III-5 donne un exemple simple de requête.

Tableau III-5 : Exemple simple de requête GDD_Q.

?n : Choice : NODE()←REL()←$n()
Cette requêtte retourne tous les nœuds *$n* en relation directe avec le nœud « Choice ». On cherche les nœud *$n* (?n) qui peuvent être trouvés par le chemin (NODE()←REL()←$n()) en partant de la racine « Choice ». Le chemin commence à « Choice ». « Choice » correspond à NODE(). On passe ensuite les relations qui ont pour cible « Choice » (←REL()). Il n'y a pas de condition particulière sur les relations à franchir (pas de paramètre à REL). On sélectionne ensuite les nœuds qui sont source (←$n()) de ces relations.

La grammaire de GDD_Q est fournie dans le Tableau III-6.

Tableau III-6 : Grammaire du langage de requête GDD_Q.

Symbole	Traduction	Commentaire
GDD_Q	'?' L_VARS ':' ROOT ':' PATH	Une requête sur le GDD pose les variables à trouver, la racine à partir de laquelle la requête est posée et le chemin à suivre.
L_VARS	VAR | VAR ',' L_VARS	
VAR	*String*	Les noms de variables sont des chaînes de caractères sans séparateurs.
ROOT	Identifant de noeud GDD	Un noeud du GDD qui servira de racine pour la requête.
PATH	NODE | NODE AR L_REL AR PATH	Un chemin est une suite de noeuds et de relations. Le sens de parcours est donné par AR (pour ARROW, flèche)
AR	'←' | '→'	Sens de parcours.
L_REL	REL | REL '>' L_REL | REL '<' L_REL	Si les nœuds entre les relations n'ont aucune importance pour la définition du chemin. Il est possible de ne spécifier que des relations séparées par '<' ou '>'

		(selon le sens de parcours). Le parcours se fait alors quelque soit les caractéristiques du nœud.
REL	'REL' '(' COND ')' ['*']	On peut poser des conditions sur une relation pour la franchir. Le symbole '*' peut être utilisé pour spécifier qu'on passe un nombre indéterminé de relations vérifiant COND.
NODE	NODE_NAME '(' COND ')'	On peut poser des conditions sur un nœud pour le franchir.
NODE_NAME	'$' VAR | 'NODE'	Un nœud peut être identifié à une variable ou être anonyme. Dans le premier cas, il sera renvoyé dans la réponse si lui et les variables descendantes sont franchis.
COND	Expression booléenne	Une expression booléenne portant sur les attributs de l'objet (relation ou nœud). Il est possible de faire appel à des fonctions extérieures.

Des exemples de requêtes GDD_Q plus complexes sont donnés dans le Tableau III-7.

Tableau III-7 : Exemples de requêtes GDD_Q. Seul le chemin (PATH) est explicitement décrit.

NODE()←REL(type~=GDD_specialization)*<REL(type~=GDD_implementation)←$n()
Trouve les implémentations $n d'un nœud donné (en racine). Le chemin indique de parcourir les relations de type spécialisation ou sous type de spécialisation (~=) jusqu'à trouver une relation d'implémentation. L'implémentation trouvée est ajoutée dans la variable. A la fin, la variable n contient la liste des implémentations demandées.
NODE()→REL(type~=GDD_specialization && type!~= GDD_extension)* < REL(type~=GDD_specialization)*<REL(type~=GDD_implementation)←$n()
Trouve les implémentations équivalentes à un noeud donné (en racine). On parcourt les relations de spécialisation qui ne sont pas des extensions (qui sont tels que la racine est une de leur spécialisation « ->REL(type~=GDD_specialization && type!~= GDD_extension)* »). On cherche ensuite les implémentations de ce nœud.
NODE()←REL()*←$n()
Donne tous les nœuds du GDD.
NODE()←REL()*<REL(type~=GDD_implementation) ←$n(ptf ~= Ptf_TK)
Donne tous les systèmes compatibles avec la plate-forme technologique TK.
NODE()←REL()*<REL(type~=GDD_implementation) ←$n(ptf == Ptf_TK)
Donne tous les systèmes explicitement prévus pour la plate-forme technologique TK. (== au lieu de ~=).
NODE()→REL(type~=GDD_specialization)*→$n()
Donne les nœuds de niveau C&T correspondant à un noeud donné.
NODE()←REL(type~=GDD_specialization)*<REL(type== GDD_concretization)< REL(type~=GDD_specialization)*←$a(type==GDD_AUI)←REL(type == GDD_concretization)←$c(type==GDD_CUI)←REL(type~=GDD_specialization)*<REL(type== GDD_implementation)←$f()
Pour un nœud décrit au niveau C&T, donne les AUI ($a) donnant lieu à des CUI ($c) donnant lieu à des FUI ($f) sous forme d'arbres (niveau 1 : les AUI, niveau 2 : les CUI, niveau 3, les

FUI).
NODE()→REL(type ~= GDD_composition)→$n()
Donne les nœuds qui composent un nœud donné.

Dans ces exemples, le parcours du graphe s'appuie principalement sur les propriétés des relations. Ceci a l'avantage de ne pas avoir à traiter les descriptions des nœuds, et donc de ne pas avoir besoin de descriptions détaillées dans un langage précis de ces nœuds.

Pour assurer qu'une propriété (ex : absence de navigation) sera vérifiée par les résultats trouvés, il faut s'appuyer sur un nœud dont on sait qu'il a cette propriété (ex : entrelacement sans navigation). A partir de ce nœud, il faut ne franchir que des relations qui garantissent que la propriété sera préservée. Ce sont les relations de type « est spécialisé en ». En effet, si B spécialise A, il peut être utilisé partout où A l'est, il conserve donc les propriétés de A, quelles qu'elles soient.

En tant qu'annuaire, le GDD peut être interrogé par des systèmes interactifs en quête d'IHM (par exemple). La Figure III-58 illustre le fonctionnement à l'exécution du GDD. 1). Des concepteurs enrichissent le GDD de descriptions et de systèmes interactifs. 2) Des clients émettent des requêtes GDD_Q pour retrouver des systèmes. 3) Les nœuds du GDD correspondant à la requête sont renvoyés. 4) A partir des descriptions de ces nœuds, le client essaie d'obtenir ou de générer des instances du système demandé.

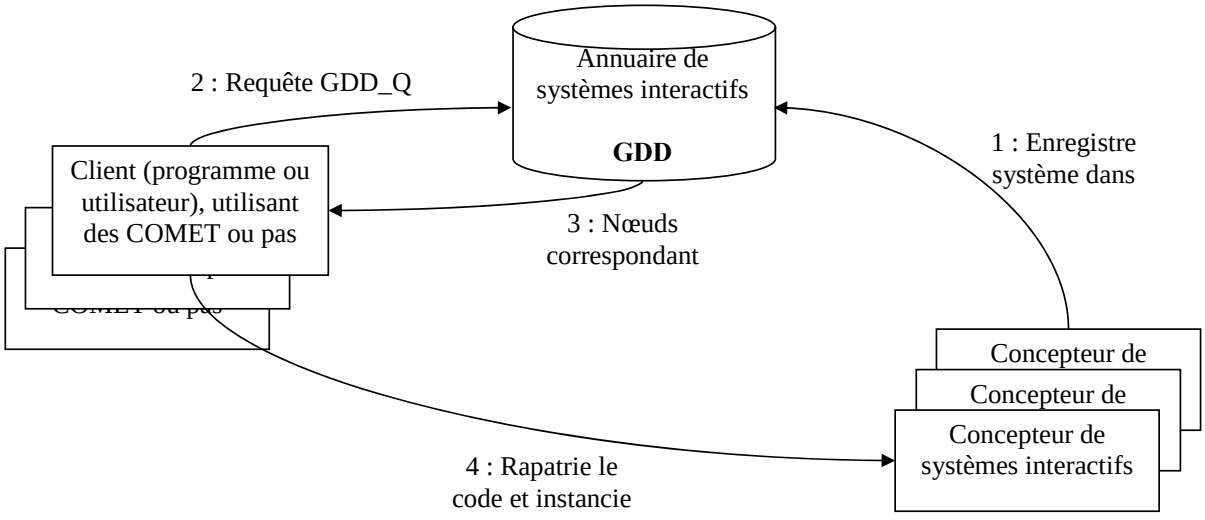


Figure III-58 : Utilisation du GDD par des systèmes interactifs, à base ou non de COMET.

III.3. Conclusion sur le GDD

Le GDD est un réseau sémantique permettant de classer les descriptions de systèmes interactifs. Il est structuré selon les quatre niveaux d'abstraction du cadre de référence CAMELEON. Le GDD peut être utilisé à la conception pour retrouver soit des spécifications de système interactif (par exemple un arbre des tâches), soit des systèmes interactifs implémentés. Lors de la phase d'exécution, le GDD peut être considéré comme un annuaire de systèmes interactifs qui, à la manière des annuaires de services, peut retrouver des systèmes selon une description qu'on en donne. Une manière, mais ça n'est pas la seule, de donner une description du système désiré, est de décrire un chemin dans le GDD, ce chemin assurant ou pas certaines propriétés sur les nœuds atteints. Le langage GDD_Q offre une solution pour la spécification de tels chemins.

Les requêtes formulées sur le GDD à l'aide de GDD_Q restent valides quelque soit les changements dans le GDD (ajout, suppression, réorganisation de nœuds) pour peu que la racine sur laquelle porte la requête ne soit pas détruite. Ainsi les mêmes requêtes GDD_Q fourniront d'autant plus de résultats que le GDD est enrichi. Une application qui utiliserait les résultats de requêtes GDD_Q pour enrichir son IHM pourrait donc être d'autant plus enrichie que le GDD est fourni. Ceci est assuré sans avoir à changer une seule ligne de code du système interactif.

La qualité des résultats fournis dépendra bien entendu de la qualité du GDD lui-même, en termes des systèmes qui y sont stockés et de la structuration du réseau.

IV. CSS++ : Un langage de style pour la plasticité

L'architecture logicielle des COMET, couplée au GDD, offre un grand nombre de possibilités de représentation de chaque COMET et, par extension, un grand nombre de possibilités de présentation pour un système interactif. Le problème se pose donc de savoir comment naviguer dans cet espace de possibilités. J'ai voulu donner le contrôle des choix de présentation au concepteur. Ce contrôle s'effectue via des styles.

Les langages de style pour les IHM sont principalement utilisés dans le monde WEB avec CSS et, dans une moindre mesure, XSLT. Le langage CSS est très largement utilisé, y compris par des concepteurs amateurs. C'est pour cela que j'ai choisi de me baser sur CSS pour définir CSS++, un langage de style pour la plasticité des IHM. Il est applicable aux graphes de COMET. Il peut faire appel au GDD. Il n'est en effet pas envisageable de reprendre CSS tel quel, notamment le sous langage des sélecteurs, pour plusieurs raisons :

- CSS fait la supposition que les sélecteurs s'appliquent sur un arbre, or les COMET sont structurées en DAG.
- Les COMET sont structurées en graphes hiérarchiques. Il faut donc un moyen d'accéder aux graphes contenus dans des nœuds.
- Les COMET superposent trois types de graphes : LC, LM et PM. Il faut pouvoir passer de l'un à l'autre.
- En CSS, les actions exprimables sont limitées à des modifications d'attributs par des valeurs simples. Avec les COMET et la plasticité, il est nécessaire de pouvoir faire appel au GDD et de définir des actions plus complexes, du type règles ECA (Evènement Condition Action) ou appel de fonction. Un exemple de règle ECA dans Sedan-Bouillon peut être « si l'utilisateur U s'identifie sur une plate-forme (évènement) et qu'il est déjà identifié sur une autre plate-forme (condition) alors lui montrer la méta-IHM de redistribution des espaces de dialogue sur les plates-formes ».

D'autres limites de CSS, non directement liées aux COMET, m'ont poussé à étendre le langage :

- Les sélecteurs manipulent des ensembles, mais seul l'opérateur d'union ensembliste est présent. Les opérateurs de différence et d'intersection ensemblistes ont été ajoutés à CSS++.
- La notion de négation n'est pas définie dans CSS. CSS3 propose certes la pseudo classe not, mais elle ne porte que sur un sélecteur simple. Ainsi, il n'est par exemple pas possible d'exprimer des sélecteurs tels que : « Les choix qui n'ont pas de fils ayant des descendants ».
- Il n'est pas possible de sélectionner un nœud autrement que par ses attributs et ses ancêtres. En particulier, il n'est pas possible de sélectionner un nœud par rapport à ses descendants. Or, il est souhaitable de pouvoir exprimer des sélecteurs tels que : « les Containers qui contiennent (ont pour descendants) des images » ou « les choix qui portent sur des éléments importants ».

Malgré ces limitations, j'ai tenu à conserver une grammaire proche de CSS : c'est une grammaire simple à comprendre et à écrire. Pour preuve, j'en tiens le large usage de CSS par des concepteurs, novices ou pas, du monde WEB. XPath ne souffre pas des limitations de CSS mais est plus verbeux. Hormis les pseudo sélecteurs, CSS++ reste compatible avec CSS : une règle CSS sans pseudo sélecteur reste valide en CSS++.

Le langage CSS++ n'est pas exclusivement lié aux COMET. Il peut s'appliquer à n'importe quelle structure de graphe hiérarchique. Toutefois, le seul interpréteur existant a été conçu pour être appliqué à des graphes de COMET. La Figure III-59 illustre le fonctionnement général de cet interpréteur à l'exécution. L'interpréteur CSS++ lit un ou plusieurs documents CSS++ (1). Les règles sont ensuite appliquées sur le graphe de COMET (2). L'application des règles peut faire appel au GDD (2').

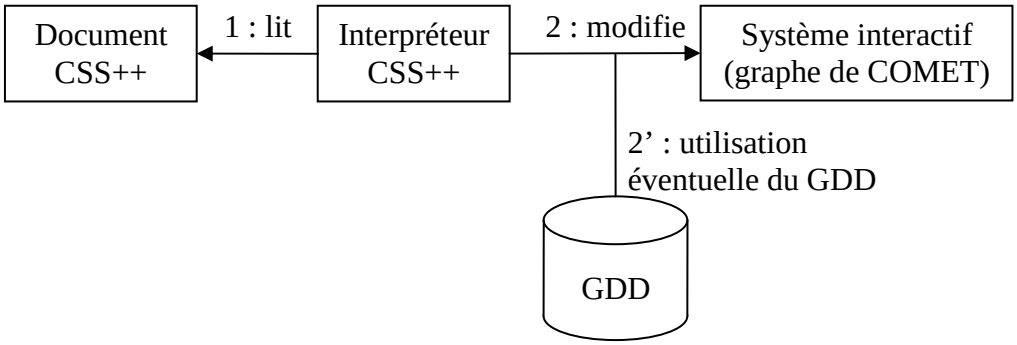


Figure III-59 : CSS++ utilisé avec le GDD et les COMET.

La Figure III-60 montre un exemple de document CSS++.

```
Implem : NODE()←REL(type~=GDD_specialization)*
<REL(type~=GDD_implementation)←$n() ;
Compo : NODE()→REL(type == GDD_composition)→$n() ;
__RULES__
CometContainer CometContainer {
  border : none;
}
CometInterleaving \! CometInterleaving / {
  background : blue;
  GDD_fct : Implem( MenuMonospaceInterleaving );
}
```

Déclaration de fonctions GDD_Q pour une utilisation ultérieure, dans les règles CSS++.

Déclaration de règles CSS++, cette partie est semblable aux déclarations CSS.

Figure III-60 : Exemple de document CSS++.

Pour palper la puissance d'expression et la concision des sélecteurs CSS++, je propose de comparer sur un panel d'exemples les langages XPath1 et CSS++ (Tableau III-8).

Tableau III-8 : Comparatif de XPath1 et CSS++.

Objectif	XPath 1	CSS++
Les fils de A qui ne sont pas des B	A/*[name() !='B']	A > !B
Les fils de A qui ne sont ni des B ni des C	A/*[name() !='B' and name() != 'C']	A > !(B, C)
Les B fils de A	A/B	A > B
Les B descendants de A	A//B	A B
Les A qui ont des B pour descendants	B/ancestor::A	A \B/

Les B qui n'ont pas de A pour ancêtres	B[not(ancestor::A)]	#Root >>!A>> B
Les ancêtres de A	A/ancestor::*	* \A/
Les A qui n'ont pas de B pour fils	A[not(B)]	A \!> B/
Les A qui n'ont que des B pour fils	A[not(*[name() != 'B'])]	A \!> !B/
Les C qui ont au moins un A et un B pour ancêtre	C[ancestor::A][ancestor::B]	(A C) ; (B C)
Tous les B sauf ceux qui sont fils d'un A	B[not(./parent::B/parent::A)] (Valide seulement sur un arbre.)	B – (A > B)
Les A qui ont plus de 4 fils	A[number(*) > 4]	A \MATH #(>*) > 4/
Les A qui ont plus de B que de C pour fils	A[count(B) > count(C)]	A \MATH #(>B) > #(>C)/
Les E qui ont un C ou un D pour père. C et D ayant un A ou un B pour ancêtre	A///C/E | A///D/E | B//C/E | B//D/E ou *[name()='A' or name() = 'B']//*[name()='C' or name()='D']/E	(A,B) (C,D) > E
Les A plus proches ancêtres d'un B	B/ancestor::A[0] (Valide seulement si la liste des ancêtres est classée selon le nombre de génération.)	A \>>!A>> B/
Les A plus lointains ancêtres d'un B	A[not(./parent::A/ancestor::A)] [.//B] (Valide seulement sur un arbre.)	(A \B/) – (A A)
Les A qui ont une majorité absolue de B pour fils	A[count(B) > count(*[name() != 'B'])]	A \MATH #(>B) > #(>!B)/
Les B descendants de A qui ne sont pas séparés de ce A par un C	∅	A >> !C >> B

La grammaire CSS++ est présentée dans le Tableau III-9.

Tableau III-9 : Structuration générales de la grammaire de CSS++.

Symbole	Traduction	Commentaires
CSS++	[LF_GDD_Q '__RULES__'] RULES	Un « document » CSS++ contient un ensemble de règles CSS++ éventuellement précédées de définitions de fonctions GDD_Q.
LF_GDD_Q	F_GDD_Q | F_GDD_Q LF_GDD_Q	Une liste de fonctions GDD_Q est composée de fonctions GDD_Q.
F_GDD_Q	*string* ':' *string* ';'	Une fonction GDD_Q est définie par un nom et un chemin. Ex : Compo : NODE()→REL(type ==

		GDD_composition)→$n()
RULES	RULE | RULE RULES	
RULE	SELECTOR '{' ACTIONS '}'	Une règle est composée d'un sélecteur et d'une liste d'actions. Ex : H P {font : bolder ; color : grey ;} Les sélecteurs sont décrits dans le Tableau III-10.
ACTIONS	ACTION | ACTION ';' ACTIONS	Les actions sont séparées par des ';'. Les actions sont décrites dans le Tableau III-11.

Tableau III-10 : Grammaire des sélecteur CSS++.

Symbole	Traduction	Commentaires
SELECTOR	SEL | SEL OP_S SELECTOR	Un sélecteur est une liste de SEL séparés par des opérateurs ensemblistes. Ex : A, B
OP_H	',' | ';' | '-'	Union, intersection et différence ensemblistes.
SEL	CLASS | CLASS OP_H SEL	Un sélecteur est une liste de CLASS séparés par des opérateurs de parcours de graphe. Ex : A > B C
OP_H	['~'] OP_H_SIMPLE	'~' indique que le parcours se fait en prenant en compte les graphes hiérarchiques. Si '~' n'est pas présent, le parcours ne rentre pas dans les graphes hiérarchiques.
OP_H_SIMPLE	' ' | '>' | '>>' SELECTOR '>>'	Descendants (' '), fils ('>') ou bien propriété sur le chemin ('>>'…'>>') Une propriété sur le chemin exprime des contraintes sur les nœuds qui peuvent être traversés. Par exemple A >>!C>>B indique les B qui sont atteignables par un A sans qu'un C les sépare. A>>(C>D), (E>F)>>B indique les B atteignables par un A tels que entre les deux il n'y a que des couples « C puis D » ou « E puis F ».
CLASS	DEF [FILTER]*	Une CLASS est composée d'une DEF suivie d'une liste, éventuellement vide, de filtres.
FILTER	'\' ['!'] ['~'] ['>'] SELECTOR '/' | '\' ['!'] ['~'] '>>' SELECTOR '>>' SELECTOR '/' | '\MATH ' L_EXPR '/'	Ne retenir que les noeuds qui ont ou n'ont pas ('!') pour fils ('>') ou descendants, au sens large ('~', rentre dans les graphes hiérarchiques) ou pas des nœuds qui vérifient SELECTOR. Si '>>', ne retenir que les nœuds qui vérifient la contrainte de chemin donnée entre

		les '>>' pour aboutir à SELECTOR. Si 'MATH', ne retenir que les nœuds qui vérifient l'expression booléenne L_EXPR.
L_EXPR	*Expression booléenne*	Expression booléenne portant sur les descendants du nœud courant. Il est possible d'utiliser des fonctions qui prennent des SELECTOR en paramètre (ex : le cardinal). Par exemple : le nombre de descendants de type A est il plus grand que le nombre de fils de type B ? (#(A) > #(>B)).
DEF	ID ['(' ['>'] SELECTOR ')']	Les SELECTOR tels qu'ils sont dans un graphe contenu dans un nœud sélectionnable par ID. SELECTOR est un chemin exprimé par rapport à la racine du graphe contenu par ID ('>')ou pas.
ID	NAME ['->' PROJ]	ID est identifié par NAME. Eventuellement ('->') on change de graphe avec PROJ.
PROJ	'LC' | '_LM_LP' | 'PMs'	Passer de l'élément courant à (aux) élément(s) correspondants dans un autre graphe (graphe de LC, graphe des LM de présentation, graphe des PMs de présentations).
NAME	['!'] NAME_SIMPLE ['[' COND ']']	Les nœuds qui vérifient NAME_SIMPLE ou bien ('!') ceux qui ne le vérifient pas. Une condition COND peut en outre porter sur le nœud.
NAME_SIMPLE	[#] *string* ['.'*string*]* | '(' SELECTOR ')'	Le nœud courant a-t-il pour marque les *string*? Ou bien ('#') donner tous les *string* (le premier) qui ont pour marques les *string* (les derniers). En cas de parenthèses, NAME_SIMPLE renvoie les nœuds qui vérifient SELECTOR.
COND	*Expression booléenne*	Expression booléenne portant sur les attributs du noeud

Tableau III-11 : Actions possibles dans les règles CSS++.

ACTION	Commentaires
att : val ;	L'attribut « att » des éléments sélectionnés prend la valeur « val ».
FCT_*fct* : param ;	Pour chaque élément sélectionné, la fonction *fct* est appelée avec le paramètre « param ».
ECA : *Event, Condition, Action* ;	Une règle ECA est posée sur l'élément sélectionné. Si

	l'événement « event » se produit (cet événement n'est pas forcément lié à l'élement), sous condition que « cond » soit vraie, alors « Action » est déclenchée. « Action » est un mini programme.
GDD_fct : *Function* (*root*) ;	Un appel est fait au GDD pour modifier le type de présentation des éléments sélectionnés. Si l'élément est un LC ou un LM de présentation, alors tous les PM sont modifiés. Si c'est un PM, seul celui-ci est modifié. « Fonction » est une fonction sur le GDD, c'est-à-dire un nom désignant une requête GDD_Q. « Root » est la racine à partir de laquelle la requête GDD_Q sera exécutée.

Pour exploiter efficacement CSS++ avec les COMET, chaque élément de COMET (LC, LM, PM) est marqué de son nom et ses types. En outre, les marques d'un LC se retrouvent dans ses LM et celles d'un LM dans ses PM. La Figure III-61 illustre ce principe sur une COMET Text nommée « nom » (types : CometElement, LC, CometText). Elle a un LM de présentation nommé « nom_LM_LP » (types : CometElement, LM, CometText_LM_LP) et des PM de présentation, notamment un nommé « nom_PM_P_3 » (types : CometElement, PM, PM_P_TK, CometText_PM_P_TK).

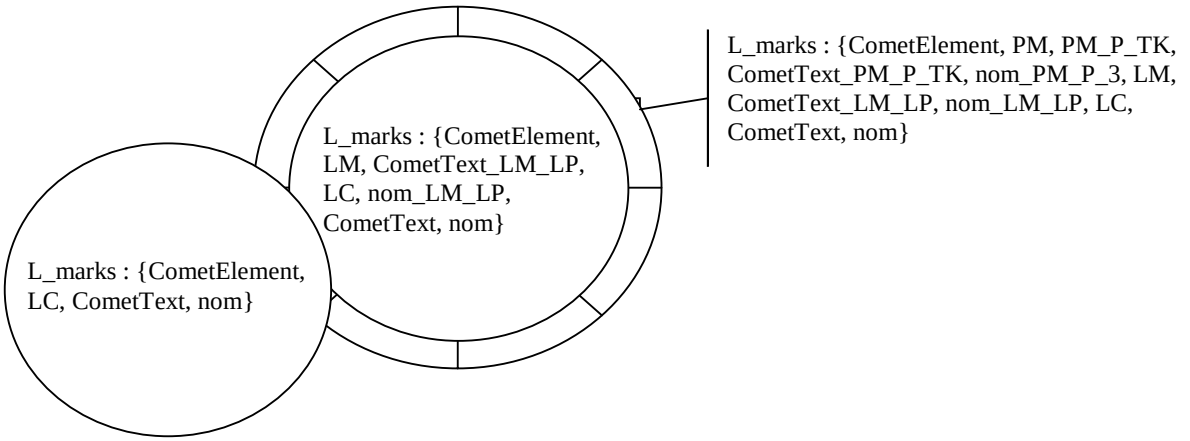


Figure III-61 : Exemple d'utilisation des marqueurs.

Ainsi, une COMET est sélectionnée si elle contient une marque qui correspond au nom recherché. Par exemple « * CometChoice » sélectionnera tous les choix qui ont un ancêtre. Selon que le graphe de départ est le graphe des LC, des LM ou des PM de présentation, les choix renvoyés seront des LC, des LM ou des PM. Le Tableau III-12 présente des exemples de règles CSS++ appliqués aux COMET.

Tableau III-12 : Exemples de règles CSS++.

CSS++	Commentaires
#CometChoice->PMs[nb_max_choices > 1](CometMarker) {GDD_fct : Implem(Marker_CUI_checkbox);}	Sélectionner tous les marqueurs liés à des choix multiples. Les représenter par des cases à cocher.
#CometContainer->PMs[soft_type == PHP_HTML] \>>!CometContainer>> CometSlideRemoteController/ {GDD_fct : Implem(Container_Maskable_CUI); border: solid red 2px;	Les présentations HTML de conteneurs plus proches ancêtres d'une télécommande deviennent des conteneurs masquables avec une

}
#CometCamNote_Speaker {
ECA : $obj set_preso_mode,
$m == 1,
set telec [CSS++ $obj CometSlideRemoteController] ---
C_GDD($telec, Implem, N_controller_CUI_basic_keyboard)
}
	bordure rouge de 2 pixels.
	Si le présentateur passe en mode de présentation plein écran ($m == 1) alors choisir une télécommande au clavier.

#ccn_visitor_1->PMs[soft_type == PHP_HTML] ~ CometInterleaving[nb_daughters > 8]
ou
#ccn_visitor_1->PMs[soft_type == PHP_HTML] ~ CometInterleaving \MATH #(>*) > 8/ {
GDD_fct : Implem(MenuMonospaceInterleaving);
}
	Les entrelacements HTML de la COMET gérant le visiteur 1 et ayant plus de 8 fils sont représentés par des entrelacements avec un menu donnant accès à un espace à la fois.

#CometCamNote_Speaker ~ CometActivator.important {
FCT_set_RE_activate : R(*);
}
	Les PM de COMET Activator du présentateur, marqués comme important, doivent être activés de façon redondante.

[53 Hâkon 2005] montre qu'il est possible d'employer les styles CSS pour « plastifier » des pages WEB quelconques prévues pour des ordinateurs de bureau (grands écrans) vers des pages adaptées aux petits écrans, type PDA. Cette adaptation joue principalement sur une mise en colonne des balises HTML, le réglage des tailles maximales (max_length) et le non affichage d'éléments jugés superflus. De façon similaire, mais avec la BàO COMET, il devrait être possible de « plastifier » des applications réalisées en COMET mais ceci n'a pas été vérifié. Le Tableau III-13 présente des extensions spécifiques aux COMET qui pourraient être ajoutées aux règles de [53 Hâkon 2005]. Ces règles font appel au GDD.

Tableau III-13: Règles d'adaptation possibles d'un système interactif en COMET pour cibler un petit écran.

CSS++	Commentaires

CometInterleaving – (CometInterleaving CometInterleaving){
GDD_fct : Implem(MenuMonospaceInterleaving);
}
	Les entrelacements de haut niveau se font à l'aide de menu.

#CometChoice {
GDD_fct : Implem(Choice_CUI_combobox);
}
	Tous les choix sont représentés par des combobox (c'est la représentation qui prend le moins de place)

IV.1. Conclusion sur CSS++

CSS++ est un langage de style dérivé de CSS. Une expression CSS reste valide en CSS++ modulo les pseudos sélecteurs. CSS++ étend le mécanisme des sélecteurs de CSS. D'autres langages comme XPath auraient pu être choisis. Mais je souhaitais conserver une syntaxe proche de CSS compte tenu de son large succès y compris auprès de concepteurs novices. Les sélecteurs de CSS++ se rapprochent de la puissance de XPath. L'expression de chemins conditionnés ('>>') en CSS++ semble ne pas avoir d'équivalent sous XPath, mais ceci reste à vérifier.

Du point de vue des opérations applicables sur les éléments sélectionnés, CSS++ permet de dépasser les simples redéfinitions d'attributs de CSS pour proposer des appels de fonctions, la définition de règles ECA et l'appel au GDD pour rapatrier de nouvelles présentations. L'exemple de CamNote++ avec style montre qu'il est possible par ce biais de changer profondément l'aspect d'un système interactif.

V. CONCLUSION

Mes contributions logicielles s'articulent en quatre points. L'Ecosystème offre une vue générale du système interactif dans son contexte d'usage. Il permet d'appréhender le système interactif selon plusieurs points de vue (NF, tâches, AUI, CUI, FUI) et explicite les relations existant entre ces points de vue. Chaque aspect du contexte d'usage (utilisateur, plate-forme, environnement) est également modélisé dans l'Ecosystème. Les liens entre les modèles d'utilisateurs, de plate-forme, d'environnement et du système interactif sont explicitement représentés. L'Ecosystème peut ainsi être utilisé pour définir formellement des propriétés du système interactif (ex : état de distribution, éparpillement, tâches non rendues, etc.). Il devrait être possible de modifier le système interactif via des opérations appliquées à l'Ecosystème. La modification de celui-ci se répercuterait alors sur le système qu'il représente. L'Ecosystème ne présage pas de la façon dont le système interactif est construit. Des approches IDM peuvent fournir les modèles nécessaires. Une autre façon de fournir ces modèles est d'utiliser la BàO COMET.

COMET est la fois un style d'architecture et une BàO d'interacteurs plastiques (les COMET). Une COMET peut être vue à quatre niveaux d'abstraction correspondant à quatre types d'éléments dans le modèle d'architecture: le LC représente le niveau tâche, le LM de présentation représente le niveau AUI, les PM de présentation représentent le niveau CUI, les primitives technologiques contenues dans les PM de présentation correspondent au niveau FUI. Un assemblage de COMET produit trois types de graphes : le graphe des LC correspond au modèle de tâches, le graphe des LM de présentation correspond à l'AUI et les graphes des PM de présentation correspondent à différentes CUI. Les graphes de FUI, s'ils existent, sont accessibles via les primitives technologiques contenues dans les PM de présentation.

L'architecture COMET offre la possibilité de mettre en œuvre les propriétés d'équivalence et de redondance de CARE. Pour cela, un mini langage, COMET/RE, est défini. Certaines formes de complémentarité sont en outre possibles dans l'architecture COMET.

Enfin, les COMET ont une empreinte technologique large et extensible : des PM peuvent être dynamiquement ajoutés à une COMET. Dans la BàO COMET, des PM de présentation sont disponibles dans quatre technologies : B207 (post-WIMP), S207 (vocale), TK (WIMP) et AJAX (WEB). Plusieurs technologies peuvent être utilisées simultanément. L'ajout dynamique de PM peut se faire à l'aide du GDD.

Le GDD répond au besoin de capitaliser les PM des COMET, mais aussi les systèmes interactifs et leurs IHM en général. C'est un réseau sémantique classant les systèmes interactifs suivant les quatre niveaux de CAMELEON. A la manière d'un annuaire de services, il est possible de retrouver des systèmes (ex : des PM de présentation) à l'exécution pour les intégrer dynamiquement (ex : à une COMET). Le langage GDD_Q permet d'exprimer des requêtes sur le GDD. Le lien entre les COMET et le GDD peut être réalisé grâce à un langage de style, le langage CSS++.

CSS++ est un langage de style dérivé de CSS2. Il permet au concepteur de préciser comment un graphe de COMET doit être rendu. Pour cela, CSS++ peut faire appel au GDD. CSS++ permet en outre de définir des règles d'adaptation au travers des expressions ECA (Evénement, Condition, Action).

De ces quatre contributions, seul l'Ecosystème n'a pas été implémenté. Un démonstrateur, CamNote++, permet d'illustrer la mise en œuvre des trois autres contributions : COMET, GDD et CSS++.

Démonstrateur

Ce démonstrateur a été utilisé dans les publications suivantes : [8 Balme 2004], [18 Calvary 2004], [38 Demeure 2005].

Ce chapitre présente CamNote++, un logiciel de présentation plastique. CamNote++ est implémenté en COMET. Il permet d'illustrer ce style d'architecture sur un exemple plus conséquent que la seule COMET Choice (section 1). Les styles CSS++ (section 2) et le GDD (section 3) sont également illustrés sur cet exemple. En revanche, l'écosystème ne l'est pas.

I. CamNote++ et les COMET

CamNote++ requiert l'identification de l'utilisateur. L'identification se fait par un couple <Identifiant, Mot de passe> (Figure IV-1). L'IHM dépend de la plate-forme de connexion. Si l'utilisateur est distant, connecté via un navigateur Internet au logiciel CamNote++, alors l'IHM est une PM HTML de la COMET de Log (Figure IV-1-a). Si, en revanche, l'utilisateur est local, alors la PM affichée est le PM TK (Figure IV-1-b).

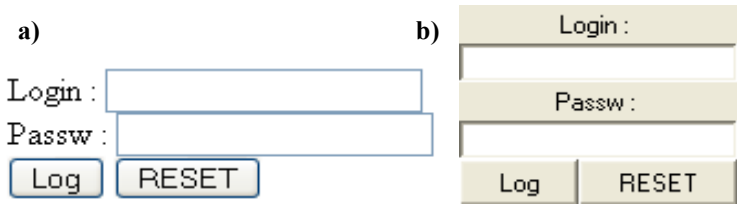


Figure IV-1 : IHM de Log. PM HTML en a ; PM TK en b.

Sur la base de l'identifiant, le système détermine le rôle de l'utilisateur : il est soit orateur, soit spectateur. Les rôles ont été préalablement définis dans un fichier de configuration. Il ne peut y avoir qu'un orateur. En revanche, le nombre de spectateurs n'est pas limité. Les tâches utilisateur dépendent du rôle de l'utilisateur (Figure IV-2-b et Figure IV-2-c). Le point commun est la visualisation, pour tous les deux, du transparent courant.

L'orateur (Figure IV-2-b) dispose à tout moment d'une télécommande pour piloter la présentation. Piloter signifie naviguer parmi les transparents, c'est-à-dire aller au premier, au dernier, au précédent, au suivant ou à un transparent de numéro donné. L'orateur contrôle aussi le mode de la présentation. Il y a deux modes : le mode présentation et le mode questions. Dans le mode présentation, l'orateur fait son exposé. Dans le mode questions, l'orateur répond aux questions posées par les spectateurs. En général, le mode questions suit le mode présentation, mais on pourrait imaginer des entrelacements, d'où le changement possible de mode à tout moment. Enfin l'orateur peut activer ou désactiver le mode plein écran pour les diapositives.

Les spectateurs (Figure IV-2-c) suivent la présentation grâce au transparent courant. Ils peuvent, à tout moment, prendre des notes, y compris inscrire de futures questions, mais ces notes restent personnelles. Lorsque le mode questions est activé, ils disposent alors d'une télécommande leur permettant de naviguer dans la présentation. Cette navigation est publique, c'est-à-dire que toutes les IHM connectées sont notifiées du changement de transparent et se rafraîchissent en conséquence. Cette fonction répond au cas classique où le spectateur dit : « Transparent 20, … ». Il se positionne alors de lui-même sur ce transparent 20.

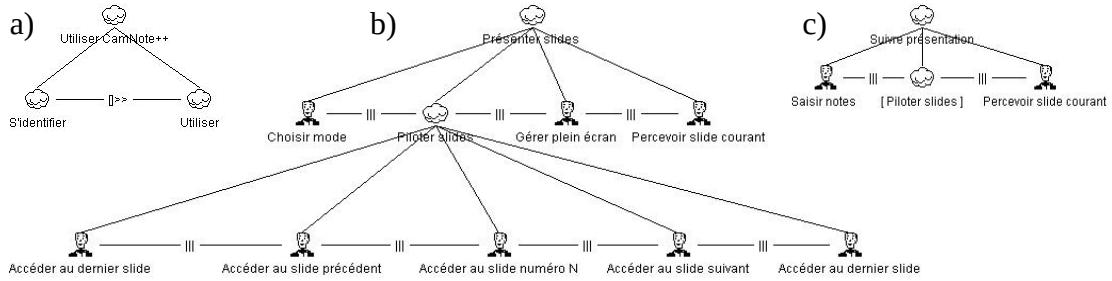


Figure IV-2 : a) Modèle des tâches permettant à l'utilisateur de s'identifier. En fonction du rôle qui lui est associé, l'utilisateur accède à : b) le modèle des tâches de l'orateur ou c) d'un spectateur.

CamNote++ est implémenté en COMET. La question première était d'identifier les COMET pertinentes pour la mise en œuvre de ce système. Il n'y a évidemment pas une solution unique. Les choix de conception ont ici été dirigés par le principe de réutilisabilité. Les tâches « Piloter slides » et « Percevoir slide courant » ont été jugées comme suffisamment générales et réutilisables pour faire l'objet de COMET à part entière.

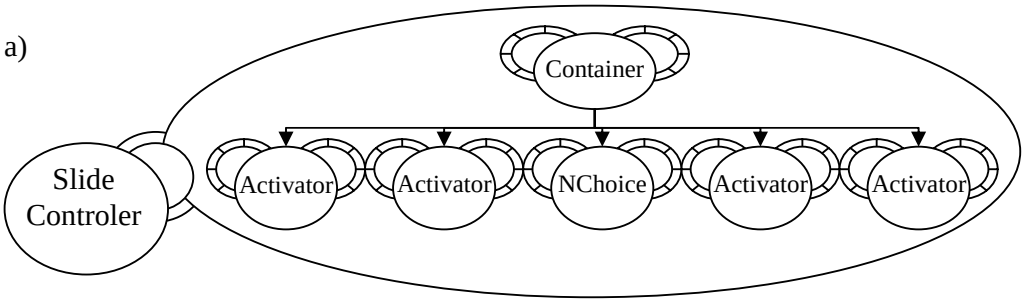
SlideControler est la COMET de pilotage du diaporama. Son API sémantique est guidée par le modèle des tâches : elle permet de naviguer entre les transparents. L'API sémantique doit permettre cette navigation. Cinq fonctions sont définies conformément au modèle de tâches : First et Last pour aller au premier et dernier transparent ; Previous et Next pour aller au précédent et au suivant ; Goto pour se positionner sur un transparent de numéro quelconque (Figure IV-3).

API sémantique SlideControler
First(): void **Previous() : void** **Goto(int) : void** **Next() : void** **Last() : void**

Figure IV-3 : API sémantique de la COMET SlideControler.

Du point de vue de l'AUI, cette COMET est un espace de dialogue. Du point de vue de la CUI, les PM peuvent être construits sur la base de COMET Activators (quatre au total pour First, Previous, Next et Last) et une NChoice pour le Goto (Figure IV-4). D'autres PM seraient envisageables, d'où la décision de laisser au niveau du PM (et non du LM voire du LC) cette décomposition. Typiquement, on pourrait imaginer des PM qui regrouperaient First et Last, d'une part, Next et Previous, d'autre part. Ces COMET « duos » seraient d'autres PM possibles. On peut aussi imaginer des PM non décomposés. La Figure IV-5 donne un exemple de rendu avec un PM sur mesure AJAX. Ce PM n'est pas décomposé en COMET.

a)



b)

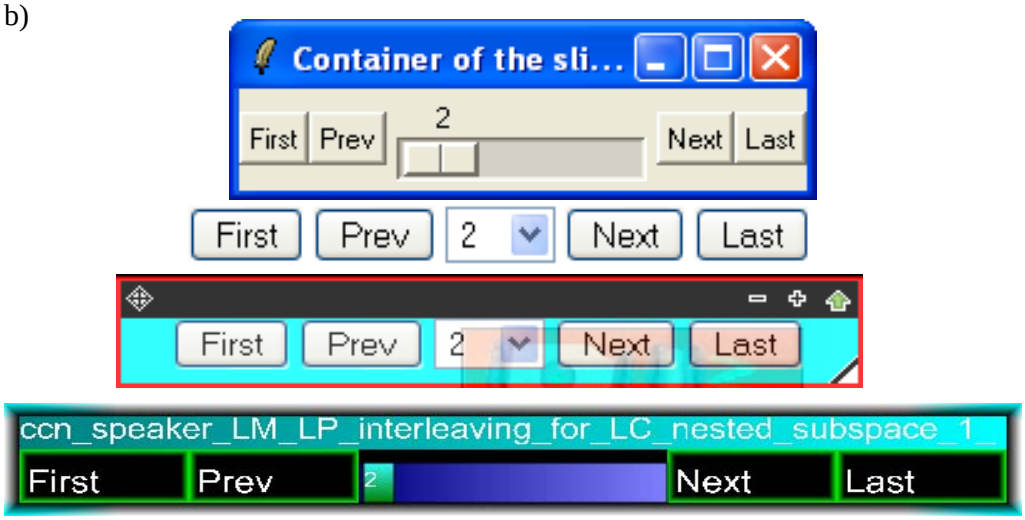


Figure IV-4 : a) Un PM de présentation pour la COMET SlideControler. b) Des rendus possibles de ce PM dans différentes technologies, avec différents styles.

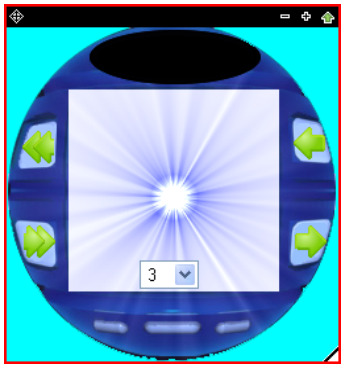


Figure IV-5 : Autre rendu de la COMET SlideControler, avec un PM « sur mesure » non décomposé en COMET.

SlideViewer est la COMET de consultation d'un transparent. Cette COMET définit une API sémantique permettant de contrôler (en lecture et écriture) la présentation et le transparent courant (Figure IV-6).

API sémantique SlideViewer
get_current_presentation(): string **set_ current_presentation(string) : void** **get_current_slide() : int** **set_current_slide(int) : bool**

Figure IV-6 : API sémantique de la COMET SlideViewer.

La COMET SlideViewer dispose de deux modèles logiques (LM). Le premier gère les aspects de présentation (à droite sur la Figure IV-7), le second gère les aspects liés au format (powerpoint, keynotes, etc.)(à gauche sur la Figure IV-7). Du point de vue de l'AUI, la COMET SlideViewer est un espace de dialogue. Du point de vue de la CUI, elle peut être représentée par un PM utilisant la structure de graphe hiérarchique et s'appuyant sur une COMET Image (Figure IV-7). Du point de vue fonctionnel, la COMET SlideViewer dispose de plusieurs PM permettant de convertir le transparent courant d'une présentation (powerpoint, keynotes, liste d'images, etc.) en image (Figure IV-7).

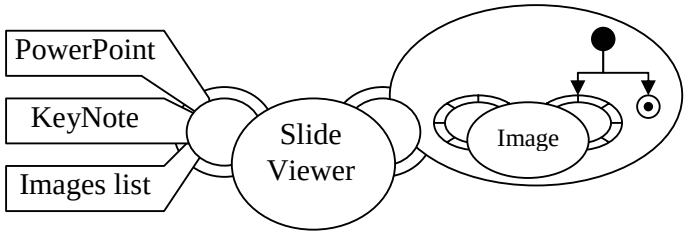


Figure IV-7 : COMET SlideViewer.

CamNote++ met en scène deux rôles : l'orateur (speaker) et le spectateur (spectator). Chaque rôle est implémenté dans une COMET : respectivement, CN_Speaker et CN_Spectator (CN pour CamNote). Les rôles ont des points communs : ils peuvent chacun, selon le mode de présentation, piloter et percevoir la présentation. Ils ont aussi des différences : l'orateur peut contrôler le mode de présentation ; le spectateur peut prendre des notes sur la présentation. Ces points communs et différences se retrouvent dans les API sémantiques (Figure IV-8).

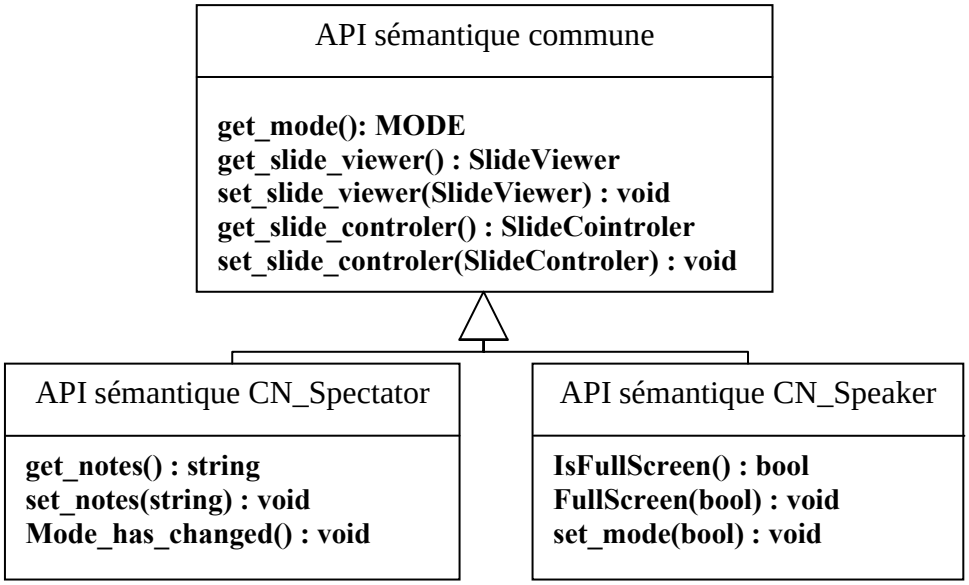


Figure IV-8 : API sémantique des COMET CN_Spectator et CN_Speaker.

Les COMET CN_Spectator et CN_Speaker incorporent une COMET SlideControler (pilotage de présentation) et sont liées à une COMET SlideViewer. La COMET CN_Spectator incorpore en plus une COMET TextSpecifyer pour la prise de notes. La COMET CN_Speaker incorpore une COMET Choice pour la spécification du mode (présentation ou questions) et une COMET Activator pour le contrôle du plein écran. Ici, l'incorporation est faite au niveau du LC : tous les LM et PM en héritent donc. Ce choix se justifie par le caractère sémantique de la décomposition (Figure IV-9 et Figure IV-10). Il suffit alors au niveau des PM de spécifier le point d'entrée dans les

PM encapsulées. Les Figure IV-9 et Figure IV-10 fixent le point d'entrée sur le PM d'entrelacement.

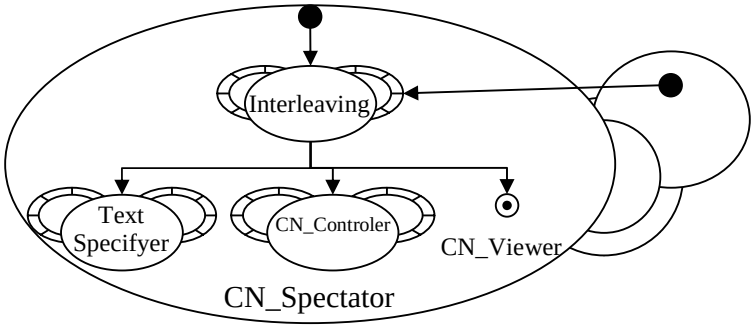


Figure IV-9 : COMET CN_Spectator.

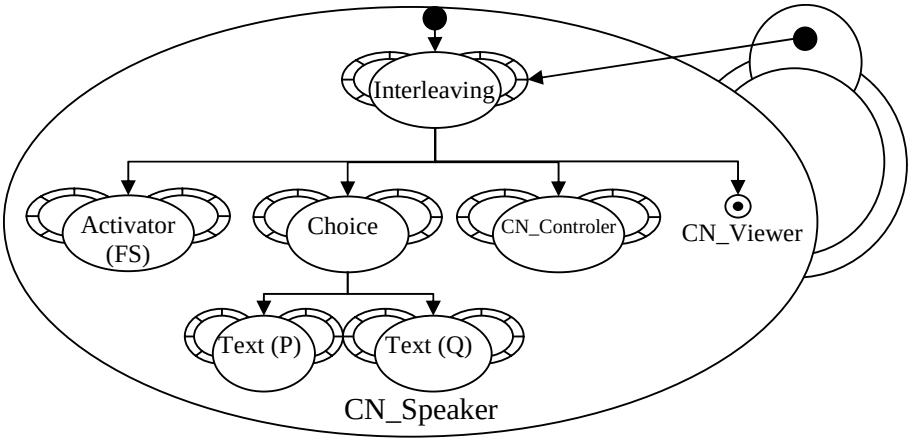


Figure IV-10 : COMET CN_Speaker.

Ces COMET étant définies, il reste à les intégrer dans la COMET CamNote++. Celle-ci s'appuie sémantiquement sur les COMET CN_Spectator, CN_Speaker et SlideViewer (Figure IV-12). CamNote++ incorpore une COMET CN_Speaker et un ensemble de COMET CN_Visitor (une par visiteur). CamNote++ définit une API sémantique permettant de gérer les visiteurs, le mode de présentation (présentation ou questions), le transparent courant et d'accéder aux COMET gérant les utilisateurs (Figure IV-11). Lorsque le mode de présentation change ou le transparent courant change, la COMET CamNote++ doit en effet assurer la mise en cohérence des différentes COMET (speaker et spectators).

API sémantique CamNote++
get_Speaker() : CN_Speaker **set_Spkeaker_log(string, string) : void** **get_Viewer() : SlideViewer** **get_Spectator (string) : CN_Visitor** **get_Spectators() : list of CN_Visitor** **set_Spectators(list of CN_Visitor) : void** **add_Spectator(string, string) : int** **sub_Spectator(string) : int** **get(string, string) : LC** **Load_presentation(string) : bool**

Figure IV-11 : API sémantique de la COMET CamNote++.

Plusieurs utilisateurs peuvent accéder à la COMET CamNote++. Ces utilisateurs doivent s'identifier à l'aide d'un login et d'un mot de passe. La BàO COMET ne permettant pas de gérer plusieurs utilisateurs, il a fallu gérer cela au niveau de l'application elle-même. J'ai décidé d'ancrer l'identification dans les PM de présentation (Figure IV-12). Aussi, ai-je dû enrichir l'API sémantique des PM pour intégrer cette identification (Figure IV-13). La gestion des différents utilisateurs se fait au niveau des PM de présentation. A un PM de présentation correspond un utilisateur particulier (mais à un utilisateur peut correspondre plusieurs PM de présentation).

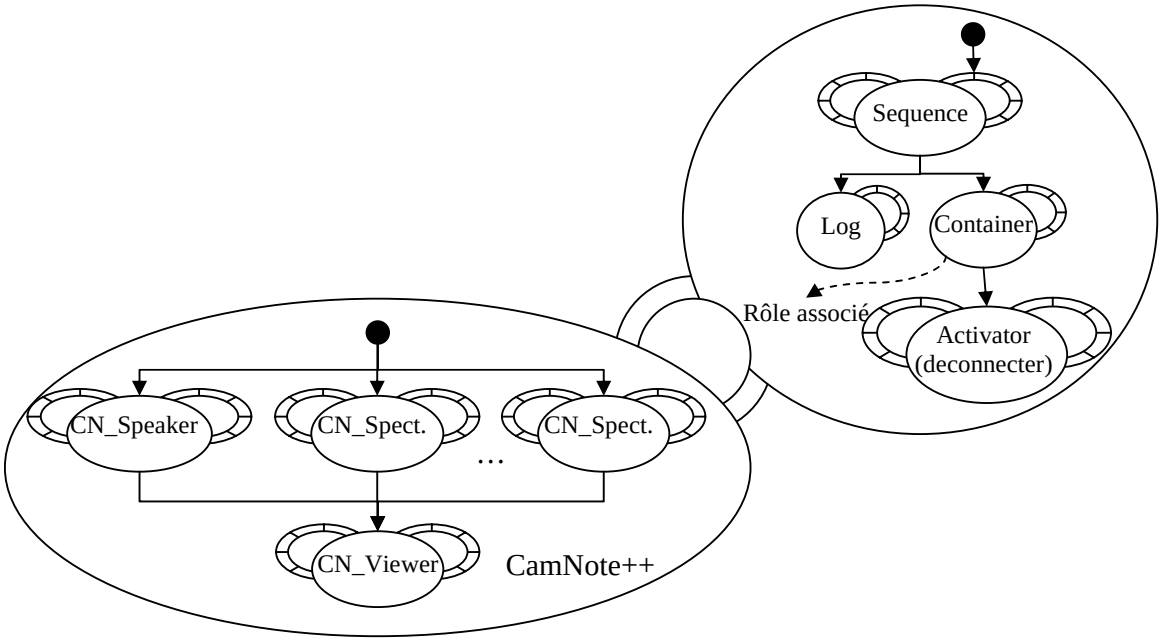


Figure IV-12 : COMET CamNote++.

API sémantique des PM de présentation de CamNote++
Connect_to(LC) : bool **Deconnect() : bool**

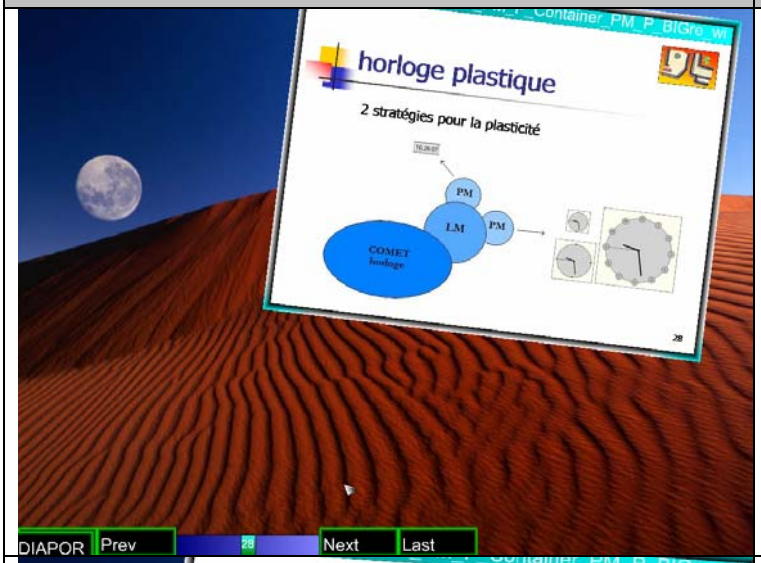
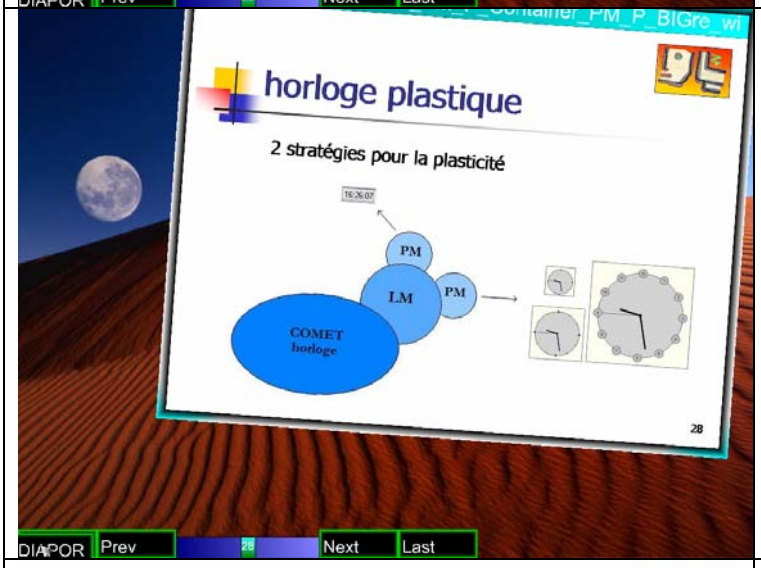
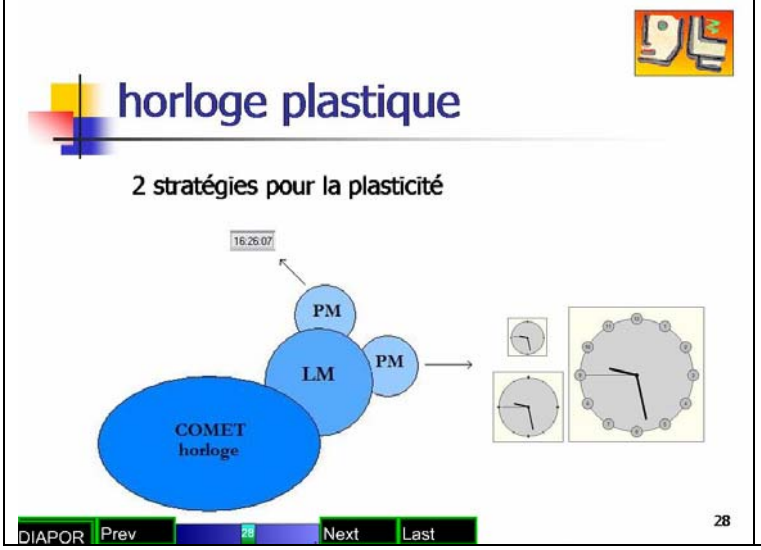
Figure IV-13 : Enrichissement de l'API sémantique des PM de présentation de la COMET CamNote++ pour permettre la connexion et la déconnexion des utilisateurs.

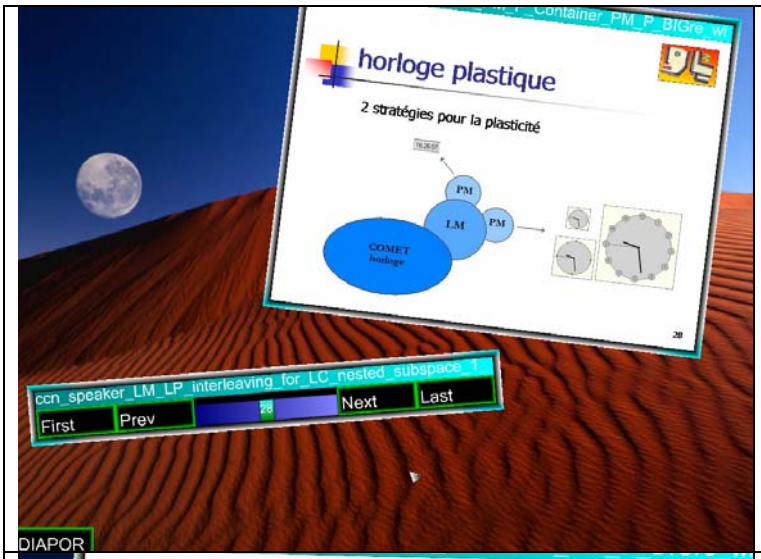
Le LC de CamNote++ gère la cohérence entre les utilisateurs. La cohérence porte sur le transparent courant et le mode de présentation. Concernant le transparent courant, la cohérence est assurée par construction, la COMET CN_Viewer étant partagée par tous les utilisateurs. En revanche, pour le mode de présentation, un traitement particulier est nécessaire : CamNote++ s'abonne à tout changement dans l'un des CN_Speaker et propage ce changement dans les CN_Spectators en mettant à jour le transparent courant.

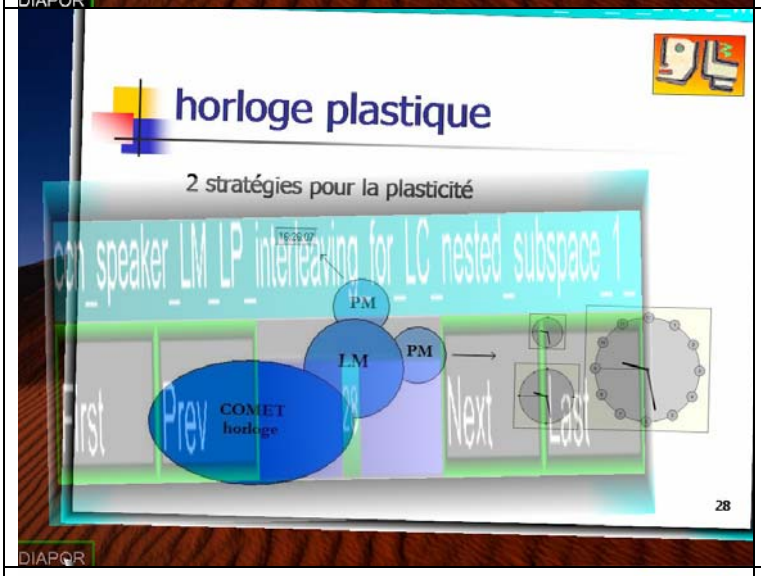
II. CamNote++ et les styles CSS++

Je montre ici quelques exemples de règles CSS++ appliquées à CamNote++ (Tableau IV-1).

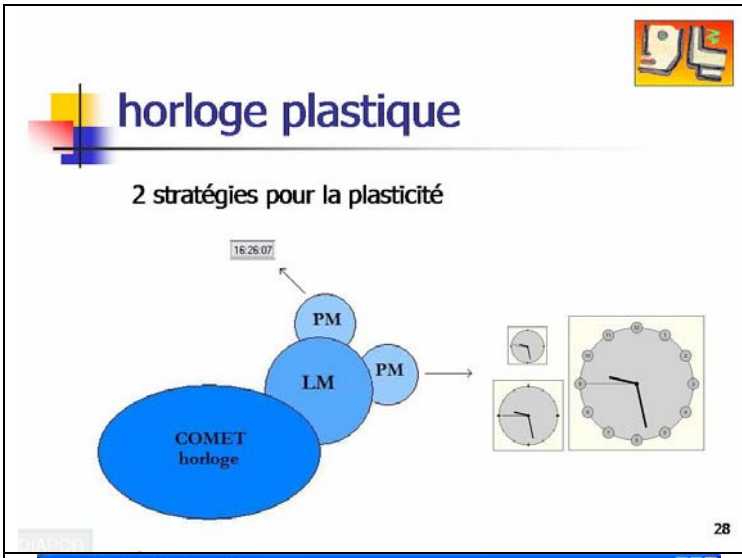
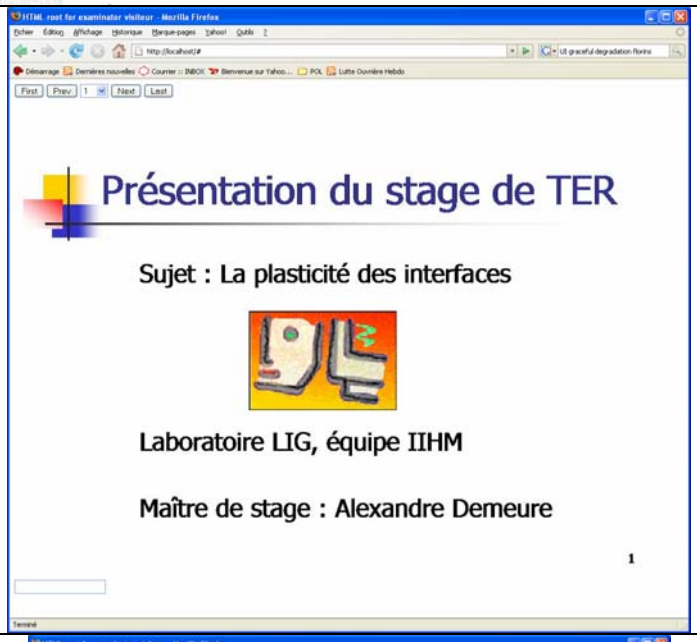
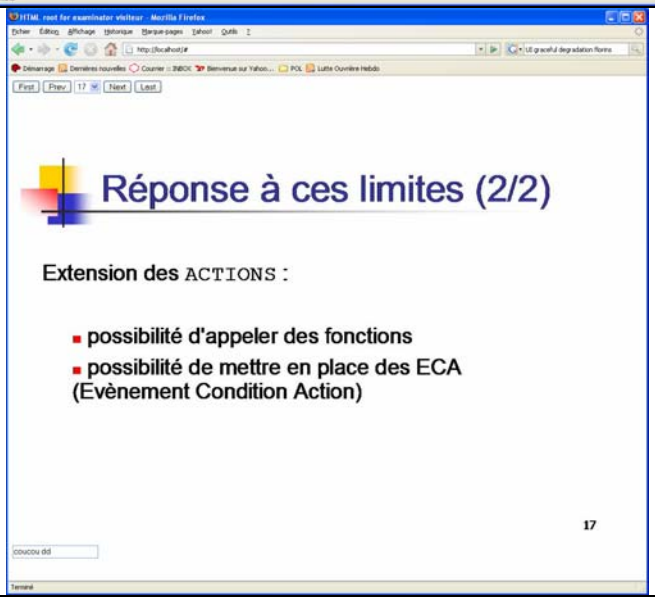
Tableau IV-1 : Application de styles CSS++ sur CamNote++.

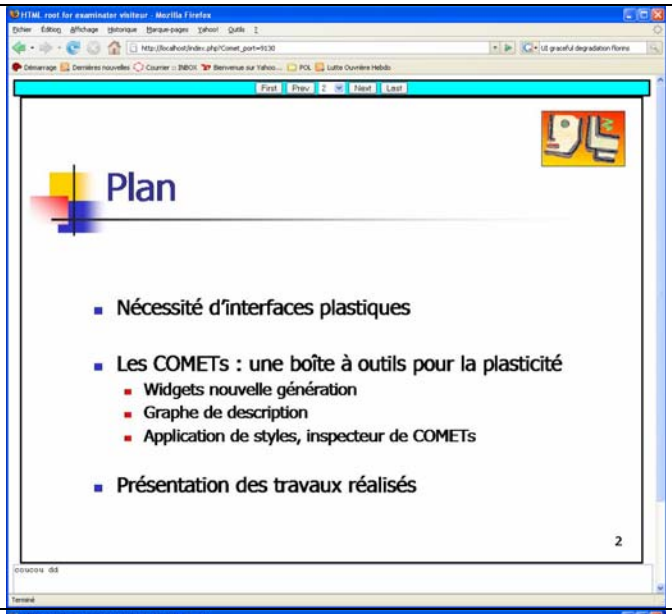
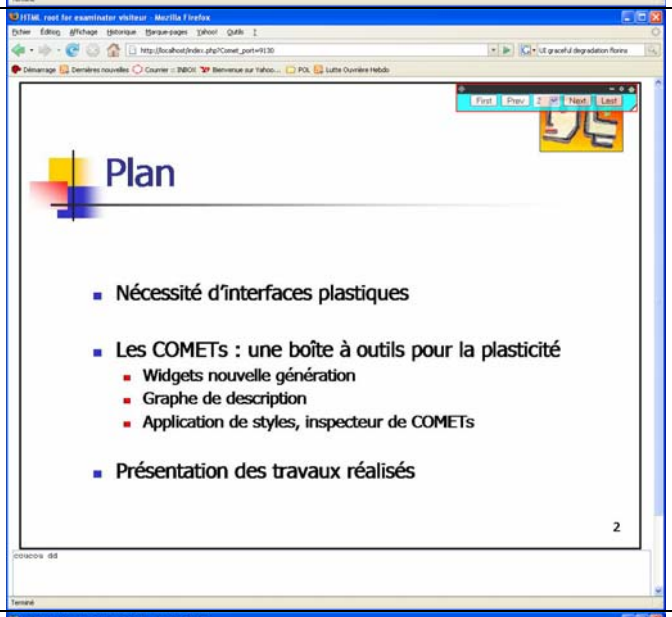
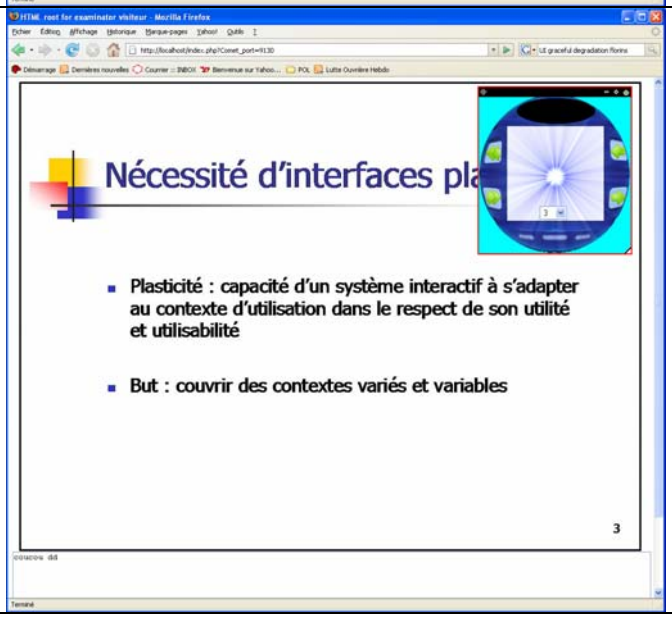
IHM	Commentaires
	Sans style appliqué, le présentateur dispose de l'IHM ci-contre (dans la technologie B207) pour mener à bien sa tâche. La télécommande est inamovible en bas à gauche et le bouton pour passer en mode plein écran (DIAPOR) recouvre partiellement la télécommande (le bouton First est masqué).
	Lorsque l'utilisateur clique sur le bouton de passage en mode plein écran (DIAPOR), le visualisateur de diapositives s'étire jusqu'à prendre tout l'écran. Cette animation est codée de base dans la version B207 de CamNote++.
	En mode plein écran, la télécommande reste en surimpression du visualisateur. Le pilotage se fait en cliquant sur les boutons de la télécommande. Celle-ci masque partiellement la diapositive courante. L'utilisateur doit, par ailleurs, viser les boutons du fait de leur petite taille.

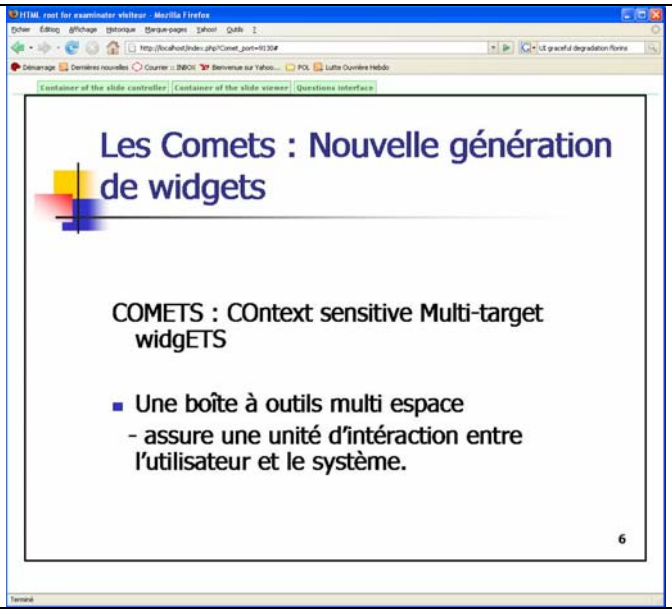
	Grâce aux styles CSS++, le conteneur de la télécommande peut devenir fenêtre et sa taille peut s'adapter à son contenu. La règle est la suivante :
	#CometCamNote_Speaker->PMs
	[soft_type==B207] CometContainer
	\>CometSlideRemoteController/ {
	GDD_fct : Implem(Container_CUI_window);
	FCT_set_method_placement:
	Window_fit_daughters ;
	}
	Le changement de la frame en fenêtre est réalisé en faisant appel au GDD (GDD_fct : Implem(Container_CUI_window))

	Lorsque le présentateur passe en mode plein écran, une animation est déclenchée. Le conteneur de la télécommande s'étire en plein écran tout en devenant progressivement transparent. Cette animation est régie par une règle ECA posée au niveau de la COMET speaker:
	# CometCamNote_Speaker->PMs
	[soft_type==B207] {
	ECA : ccn_speaker set_preso_mode,
	$m==1, ***// le nouveau mode est plein écran (1)***
	ACTION *// non détaillée car trop volumineuse*

	Une fois l'animation de la télécommande terminée, une autre animation prend place pour faire apparaître un clavier en surimpression (fade-in fade-out). Cette animation symbolise le passage de la télécommande version graphique à une télécommande clavier. Dans la partie ACTION de la règle ECA précédente, cela se traduisait par : C_B_splash($slide, $img_clavier, 1500)

<table>
<tr>
<td> 
</td>
<td>Le présentateur peut ensuite piloter la présentation en utilisant le clavier. On note que le bouton de passage en mode plein écran est devenu translucide, ce qui est moins gênant.
Dans la partie ACTION de la règle ECA précédente, cela se traduit par :
C_B_fade_bt ($transfo, $bt, 1, 0.1)</td>
</tr>
<tr>
<td> 
</td>
<td>Pour un spectateur qui utiliserait une interface WEB, l’IHM sans style serait celle ci-contre. La télécommande (en mode questions) est composée de boutons et d’un menu déroulant. Elle est placée en haut de la page. En dessous vient l’image de la diapositive courante. Puis, tout en bas, est affiché un champ texte pour la saisie des notes.</td>
</tr>
<tr>
<td> 
</td>
<td>L’utilisateur peut saisir des notes pendant que les transparents défilent.</td>
</tr>
</table>

<table>
<tr>
<td> 
</td>
<td>Grâce aux styles, la diapositive et la télécommande peuvent être encadrées. La télécommande adopte un fond cyan. La zone de note passe d'un court champ texte à une zone de texte étirée sur toute la largeur de la page. La règle de transformation de la zone de texte est la suivante :
#CometCamNote_ExaminatorQuestions->PMs CometSpecifyer {
GDD_fct :
Implem(Specifyer_basic_CUI_textarea);
width: 100%;
height: 200px;
}</td>
</tr>
<tr>
<td> 
</td>
<td>Pour que la télécommande prenne moins de place en haut de l'écran, une solution est de passer son conteneur en fenêtre. La règle est la suivante.
#CometCamNote_Examinator->PMs[soft_type == PHP_HTML] CometContainer \>CometSlideRemoteController/ {
// Appel au GDD pour trouver des PM de fenêtre
GDD_fct : Implem(Container_CUI_window);
FCT_set_width: 300; // Largeur de la fenêtre
FCT_set_height: 46; // Hauteur de la fenêtre
border: solid red 2px; // Bordure de la fenêtre
}</td>
</tr>
<tr>
<td> 
</td>
<td>Le spectateur peut préférer une télécommande plus originale.
ccn_cr_visiteur->PMs[soft_type == PHP_HTML] CometSlideRemoteController {
GDD_fct :
Implem(N_controller_CUI_basic_gfx_skinnable_simple);
}</td>
</tr>
</table>



Enfin, le visiteur peut accepter de ne pas tout avoir à la fois. Il peut donc changer l'entrelacement en colonne par un entrelacement en menu. La règle est la suivante :

```
ccn_cr_visiteur->PMs[soft_type == PHP_HTML]
CometInterleaving {
  GDD_fct :
        Implem(ScrollableMonospaceInterleaving);
}
```

III. CamNote++ et le GDD

CamNote++ est placé dans le GDD. Cette ventilation l'ouvre sur des implémentations autres que les COMET. Il suffit, en effet, d'insérer le nœud CamNote++ dans le niveau C&T du GDD (cf l'encadré « CamNote++ » à gauche de la figure Figure IV-14 dans le niveau C&T) et de permettre des concrétisations autres que les COMET (cf la relation « Concretizes » à gauche de la Figure IV-14 donnant lieu à CamNote++AUI_1 et 2, jusqu'à CamNote++ Swing et TK au niveau FUI). Bien entendu, la décomposition en C&T y est aussi placée (cf l'encadré LC_CamNote++ qui spécialise CamNote++, est composé de LM, etc.). Pour des raisons de lisibilité, l'intégralité du réseau sémantique n'est pas donnée. La Figure IV-14 en est un extrait.

Outre la réutilisabilité, l'intérêt de placer CamNote++ dans le GDD est de pouvoir passer d'une version COMET à une version non COMET et inversement. Cela peut se révéler pertinent lors de la migration d'une plate-forme à une autre pour trouver une version compatible des contraintes de la plate-forme cible. Par exemple, les COMET peuvent ne pas fonctionner sur un dispositif particulier tel qu'une montre, mais une version de CamNote++ peut avoir été conçue pour ce dispositif particulier. La Figure IV-14 montre que différentes versions peuvent exister en COMET et non COMET. Par exemple, il existe deux versions de CamNote++ COMET pour AJAX (PM1 et PM2) et deux versions TK et SWING sans COMET. Les descriptions sont organisées selon le niveau d'abstraction conformément au GDD.

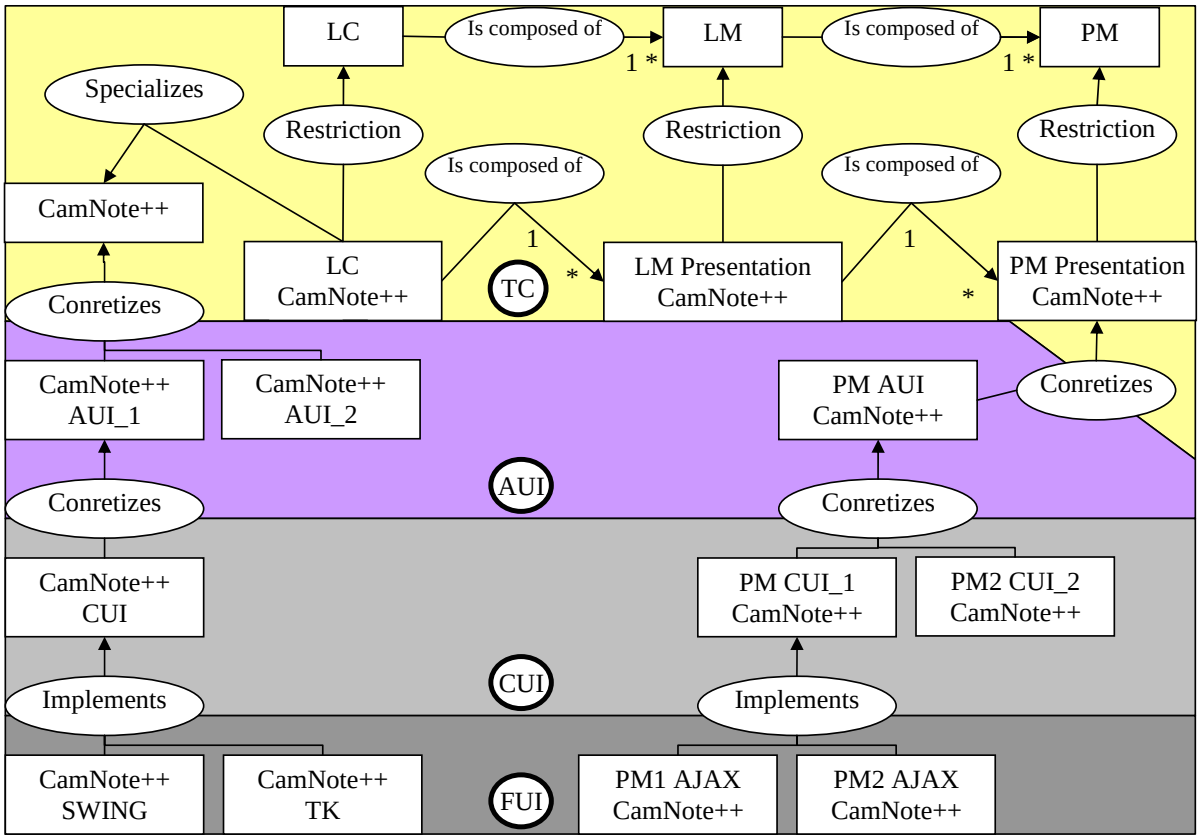


Figure IV-14 : CamNote++ dans le GDD.

IV. Conclusion

CamNote++ a permis d'illustrer mes contributions sur une application simple, opérationnelle. CamNote++ a servi, à l'équipe, d'outil de présentation à plusieurs reprises. Il illustre le concept de graphe hiérarchique, de présentation « sur mesure » et multi-technologies. Enfin CamNote++ montre que la combinaison des COMET, de CSS++ et du GDD permet de modifier profondément le rendu de l'IHM et ce pour plusieurs technologies (B207 et AJAX sont ici mis en avant, mais TK et S207 peuvent tout autant être utilisées).

A terme, la définition des règles CSS++ et les accès au GDD devraient être rendus accessibles à un utilisateur final. C'est l'objet de la Méta-IHM.

Conclusion et perspectives

Cette thèse traite du problème de la variété, variabilité et imprévisibilité du contexte d'usage. Le sujet est la proposition de modèles et outils pour la conception et l'exécution d'IHM plastiques. J'identifie trois gammes d'approches :

- l'Ingénierie Dirigée par les Modèles au cœur de laquelle se trouvent les notions clefs de modèle et de transformation : j'examine UsiXML comme représentant de ces approches ;
- les approches de types OS consistant à agir au niveau du système d'exploitation de la plate-forme cible : le gestionnaire de fenêtres Façade en est un exemple ;
- les BàO sur lesquelles les deux approches précédentes se rejoignent : mon travail se situe à ce niveau.

Les objectifs sont d'offrir des interacteurs dont le niveau sémantique est la tâche utilisateur mais qui peuvent être considérés à tout niveau d'abstraction. Les avancées en IDM sont ici mises à profit. Les interacteurs doivent par ailleurs être extensibles, notamment aptes à incorporer des présentations sur mesure. Ils doivent être transformables. Leurs rendus doivent être prévisibles et contrôlables. Enfin, leur empreinte technologique doit être variée et non exclusive.

L'examen de l'état de l'art montre qu'aucune BàO actuelle ne couvre ces requis. J'élargis mon état de l'art aux langages de transformation et aux annuaires de service pour asseoir mes contributions sur une compréhension plus large de l'existant. Cette compréhension m'a permis de poser ma vision de la plasticité et d'identifier les points à développer.

Mes contributions logicielles sont au nombre de quatre :

- L'Ecosystème est un modèle du système interactif déployé dans son contexte d'usage. L'Ecosystème permet de raisonner sur l'état du système interactif, par exemple son éparpillement ;
- Les COMET sont à la fois un style d'architecture et une BàO d'interacteurs plastiques : c'est la contribution majeure de ma thèse. Une COMET peut être vue à quatre niveaux d'abstraction donnant lieu à trois types d'éléments : le LC représente le niveau tâche, le LM de présentation représente le niveau AUI, les PM de présentation représentent le niveau CUI/FUI. Un assemblage de COMET produit trois types de graphes : le graphe des LC correspond au modèle de tâches, le graphe des LM de présentation correspond à l'AUI et les graphes des PM de présentation correspondent à différentes CUI/FUI. Les graphes sont hiérarchiques. Le style COMET répond aux requis posés. Il présente d'autres atouts, en particulier un support possible à la multimodalité. Je définis un langage COMET/RE à cette finalité. Il permettent aussi, via l'encapsulation, de régler élégamment des améliorations ergonomiques (pas seulement) ;

- Le GDD est un réseau sémantique classant les systèmes interactifs suivant leurs descriptions. Les COMET peuvent y prendre place. A la manière d'un annuaire de services, il est possible de retrouver des systèmes (ex : des PM de présentation) à l'exécution pour les intégrer dynamiquement (ex : à une COMET). Je définis le langage GDD_Q pour exprimer des requêtes sur le GDD ;
- CSS++ est un langage de style dérivé de CSS2. Il permet au concepteur de préciser comment un graphe de COMET doit être rendu. CSS++ peut faire appel au GDD. CSS++ permet en outre de définir des règles d'adaptation au travers d'expressions ECA (Evénement, Condition, Action).

De ces quatre contributions, seul l'Ecosystème n'a pas été implémenté. Le démonstrateur CamNote++ illustre la mise en œuvre des trois contributions : COMET, GDD et CSS++.

En perspectives, je souhaite, à court terme, valider le concept de COMET. Deux points de vue sont adoptables : celui du concepteur et celui de l'utilisateur final. Pour le concepteur, il s'agira de mesurer la facilité de compréhension du modèle conceptuel et implémentationnel. Pour l'utilisateur final, il faudra explorer différentes formes de plasticité en présentation et contrôle. C'est le champ de la Méta-IHM. L'inspecteur de COMET ouvre la voie. Mais il est largement perfectible dans sa version actuelle.

L'Ecosystème n'est pas implémenté. En revanche, des travaux proches sont menés à Hasselt sur le sujet [REF MoDIE…]. Il serait intéressant de comparer plus finement les approches et les propositions [25 Clerckx 2004] [74 Luyten 2006] [104 Vermeulen 2007].

A plus long terme, je souhaite intégrer mes travaux à des générateurs d'IHM. Les modèles stockés dans le GDD pourraient être exploités pour générer des PM. Une autre voie est l'écriture de transformations IDM ciblant la BàO COMET. Pour cela, les descriptions du GDD devront être affinées. Un langage de description de CUI devrait notamment être trouvé pour décrire tous les aspects graphiques (géométrie, couleurs, fontes, etc.).

Il serait aussi intéressant de pouvoir intégrer aux COMET des modèles de comportement des IHM. Des travaux existent sur le sujet [14 Blanch 2005] [5 Appert 2006]. De même, il serait intéressant d'explorer de nouvelles façons de gérer la multitude des présentations possibles (autrement que par CSS++). Là encore, des travaux existent [48 Gajos 2004] [49 Gajos 2005]. Ils fournissent des pistes intéressantes.

Pour finir, je récapitule mes publications en montrant leur couverture par rapport à mes contributions. Certaines en couvrent plusieurs.

- **ECOSYSTEME :** [39-40 Demeure 2005], [9 Balme 2006], [19 Calvary 2006], [91 Sottet 2006], [29 Coutaz 2007], [92 Sottet 2007].
- **COMET** : [33 Daassi 2003], [17-18 Calvary 2004], [37 Demeure 2004], [20 Calvary 2005], [41-42 Demeure 2006].
- **GDD** : [34 Demeure 2002], [35-36 Demeure 2003], [8 Balme 2004], [18 Calvary 2004], [41 Demeure 2006].
- **CamNote** : [8 Balme 2004], [18 Calvary 2004], [38 Demeure 2005]

Bibliographie

1. Allen JF, ***An interval-based representation of temporal knowledge***. In International Joint Conference on Artificial Intelligence, 1981.
2. Alvin T.S. Chan, Siu-Nam Chuang, ***MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing***, IEEE Transactions on Software Engineering, vol. 29, no. 12, pp. 1072-1085, Dec., 2003.
3. ***Amigo project***, http://www.hitech-projects.com/euprojects/amigo/.
4. Appert C. and Fekete J., ***OrthoZoom scroller: 1D multi-scale navigation***. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada, April 22 - 27, 2006). R. Grinter, T. Rodden, P. Aoki, E. Cutrell, R. Jeffries, and G. Olson, Eds. CHI '06. ACM Press, New York, NY, pp. 21-30.
5. Appert C., ***Modélisation, évaluation et Génération de Techniques d'Interaction***, mémoire de thèse, Université Paris-Sud XI, Mai 2007.
6. ***ATL***, http://www.eclipse.org/m2m/atl/.
7. Balme L., Demeure A., Calvary G., J. Coutaz, ***Sedan-Bouillon: A Plastic Web Site***. Plastic Services for Mobile Devices (PSMD), Workshop held in conjunction with Interact'05, Rome, 12 Septembre 2005.
8. Balme L., Demeure A., Barralon N., Coutaz J., Calvary G., ***CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces***. EUSAI 2004, LNCS 3295, Markopoulos et al.novembre 2004. pp. 291-302.
9. Balme L., Demeure A., Sottet JS., Coutaz J., Calvary G., Favre JM., ***A principled MDE Framework for Plastic User Interfaces***. 1rst Workshop onMulti-channel Adaptive Context-sensitive (MAC) Systems: Building Links between Research Communities, Glasgow, 15 mai 2006.
10. Bastien J.M.C., Scapin D., ***Ergonomic Criteria for the Evaluation of Human-Computer interfaces***, Institut National de recherche en informatique et en automatique, France, 1993.
11. Baudisch P., Xie X., Wang C., and Ma W., ***Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content***. In *Proceedings of the 17th Annual ACM Symposium on User interface Software and Technology* (Santa Fe, NM, USA, October 24 - 27, 2004). UIST '04. ACM Press, New York, NY, 91-94.
12. Bederson B., Meyer J., Good L., ***Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java***, *Proc UIST* 2000.
13. Bier E.A., Stone M., Pier K., Buxton W., DeRose T., ***Toolglass and magic lenses: the see-through interface***. In Proceeding of SIGGRAPH'93 (August 1993), ACM Press, New-York USA, 1993 pp.73-80.

14. Blanch R., ***Architecture logicielle et outils pour les interfaces hommes-machines graphiques avancées***. Thèse de doctorat en sciences de l'Université Paris XI, Orsay, septembre 2005.

15. Bolt R.A., ***"Put-That-There": Voice and Gesture at the Graphics Interface***, Computer Graphics 14(3), 1980, pp 262-270.

16. Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J., ***A Unifying Reference Framework for Multi-Target User Interfaces***, Interacting With Computers, Vol. 15/3, pp 289-308, 2003.

17. Calvary G., Coutaz J., Daassi O., Balme L., Demeure A., ***Towards a new generation of widgets for supporting software plasticity: the 'comet'***. EHCI-DSVIS'04, The 9th IFIP Working Conference on Engineering for Human-Computer Interaction, Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems, Bastide, R., Palanque, P., Roth, J. (Eds), Lecture Notes in Computer Science 3425, Springer, ISSN 0302-9743.Hamburg, Germany.11-13 Juin 2004. pp 306-323.

18. Calvary G., Demeure A., Coutaz J., Daassi O., ***Adaptation des interfaces homme-machine à leur contexte d'usage Plasticité des IHM***. La présentation d'information sur mesure, Numero Special de RIA; Paris, C. et Colineau, N. (editeurs invites). Vol 18 (4) 2004. Date de parution: Septembre 2004. , septembre. pp.577-606

19. Calvary G., Coutaz J., Daassi O., Ganneau V., Balme L., Demeure A., Sottet JS., ***Métamorphose des IHM et Plasticité : Article de synthèse***, Ergo'IA 2006, 11-13 Octobre 2006.

20. Calvary G., Daassi O., Coutaz J., Demeure A., ***Des widgets aux comets pour la Plasticité des Systèmes Interactifs***, Revue d'Interaction Homme-Machine (RIHM), Europia, Paris.Volume 6, n°1, ISSN 1289-2963. pp 33-53, 2005.

21. Capra L., Emmerich W., Mascolo C., ***CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications***, IEEE Transactions on Software Engineering, vol. 29, no. 10, pp. 929-945, Oct., 2003.

22. ***CAMELEON***, http://giove.isti.cnr.it/cameleon.html.

23. Chein M et Mugnier M.L, ***Conceptual Graphs! : Fundamental notions***, Revue d'intelligence artificielle, 6, 4, 1992, 365-406.

24. Chein M et Mugnier M.L, ***Représenter des connaissances et raisonner avec des graphes***, R.I.A, vol.10, n°1, 1996, pp 7-56.

25. Clerckx, T., Luyten, K., and Coninx, K. 2004., ***The mapping problem back and forth: customizing dynamic models while preserving consistency***, In *Proceedings of the 3rd Annual Conference on Task Models and Diagrams* (Prague, Czech Republic, November 15 - 16, 2004). TAMODIA '04, vol. 86. ACM Press, New York, NY, 33-42.

26. Coutaz J., ***Architectural Design for User Interfaces***, The Encyclopedia of Software Engineering. J. Marciniak Ed., Wiley & Sons Publ. pp. 38-49.

27. Coutaz J., Nigay L., Salber D., Blandford A., May J., Young R., ***Four Easy Pieces for Assessing the Usability of Multimodal Interaction : The CARE properties***, Proceedings of the INTERACT'95 conference, S. A. Arnesen & D. Gilmore Eds., Chapman&Hall Publ., Lillehammer, Norway, June 1995, pp. 115-120.

28. Coutaz J., ***Meta-User Interfaces for Ambient Spaces***. Invited Speaker, Tamodia'06, 8p.

29. Coutaz J., Balme L., Alvaro X., Calvary G., Demeure A., Sottet J.S., ***An MDE-SOA Approach to Support Plastic User Interfaces in Ambient Spaces***. In Proc. HCI International 2007, ERCIM SESAMI, Beijing, July 2007.

30. Crease M, Brewster S.A. and Gray P. (2000) ***Caring, sharing widgets: a toolkit of sensitive widgets***. In, 14th Annual Conference of the British HCI Group, 5-8 September 2000 British Computer Society conference series, pages pp. 257-270, Sunderland, England.

31. Crease M., ***A Toolkit Of Resource-Sensitive, Multimodal Widgets***, PhD Thesis, December 2001.

32. CSS, http://www.w3.org/Style/CSS/.

33. Daassi O., Calvary G., Coutaz J., Demeure A., Comet : ***Une nouvelle génération de "widget" pour la Plasticité des Interfaces***. IHM 2003, ACM Press. Novembre 2003. pp. 64-71.

34. Demeure A., Calvary G., ***Jeu et réalité mixte : retour d'expérience***, IHM 2002, pp. 80-87

35. Demeure A., Calvary G, Barralon N., ***Towards an Evolution Model for Supporting Plasticity of User Interfaces***.Adjunct Proceedings of HCI International 03, pp. 117-118.

36. Demeure A., Calvary G., ***Le Modèle d'évolution en Plasticité des Interfaces : Apport des Graphes Conceptuels***. IHM 2003, ACM Press.Novembre 2003. pp. 80-87.

37. Demeure A., Rouillard S., Bérard F., Calvary G., ***Requis et pistes pour les futures boîtes à outils d'interaction graphiques***. IHM'2004.

38. Demeure A., Balme L., Calvary G., ***CamNote: A Plastic Slides Viewer. Plastic*** Services for Mobile Devices (PSMD), Workshop hel in conjunction with Interact'05, Rome, 12 Septembre 2005.

39. Demeure A., Calvary G., Sottet JS., ***A Model-Driven Home Heating Control System.*** Plastic Services for Mobile Devices, PSMD, Workshop held in conjunction with Interact'05, Rome, 12 Septembre 2005.

40. Demeure A., Calvary G., Sottet JS., Vanderdonkt J., ***A Reference Model for Distributed User Interfaces***. TAsk MOdels and DIAgrams for user interface design TAMODIA'05.

41. Demeure A., Calvary G., Coutaz J., Vanderdonckt J., ***The COMETs Inspector: Towards Run Time Plasticity Control Based on a Semantic Network***, TAMODIA'2006. Hasselt, Belgium, 23-24 october, 2006.

42. Demeure A., Calvary G., Coutaz J., Vanderdonckt J., ***The Comet Inspector: Manipulating Multiple User Interface Representations Simultaneously***, In Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'06 (Bucharest, 6-8 June 2006), Chapter 13, Springer-Verlag, Berlin, 2006, pp. 167-174.

43. Farenc, C., ***Ergoval : Une méthode de structuration des règles ergonomiques permettant l'évaluation automatique d'interfaces graphiques***, thèse de doctorat en informatique, Université Toulouse 1. 1997.

44. Findlater L., McGrenere J., ***A comparison of static, adaptive, and adaptable menus***. In Proceedings of CHI'04, pages 89–96. ACM Press, 2004.

45. Florins M. and Vanderdonckt J., ***Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems***. In Proceedings of the 2004 International Conference on Intelligent User Interfaces IUI 2004 (Funchal, Madeira Island, Port., January 13-16, 2004).

46. Florins M., Montero F., Vanderdonckt J., Michotte B., ***Splitting Rules for Graceful Degradation of User Interfaces***, Proc. of 8th Int. Working Conference on Advanced Visual Interfaces AVI'2006 (Venezia, May 23-26, 2006), ACM Press, New York, 2006, pp. 59-66.

47. ***Fresco***, http://www.fresco.org/.

48. Gajos K., Weld D., ***SUPPLE: automatically generating user interfaces***, In IUI '04: Proceedings of the 9th international conference on Intelligent user interface, Funchal, Madeira, Portugal, 2004, pp 93-100.

49. Gajos K., Weld D., ***Preference elicitation for interface optimization***, In UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology, Seattle, WA, USA, 2005, pp 173-182.

50. Ganneau V., Calvary G. Demumieux R. ***Métamodèle de Règles d'Adaptation pour la Plasticité des Interfaces Homme-Machine***, IHM'07.

51. Griffiths T., Barclay P., McKirdy J., Paton N., Gray P., Kennedy J., Cooper R., Goble C., West A., Smyth M. ***Teallach: A model-based user interface development environment for object databases***. In Proceedings of UIDIS'99. IEEE Press. 86-96.

52. Grolaux D., Vanderdonckt J., Van Roy P., ***Attach me, Detach me, Assemble me like You Work***, Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2005 (Rome, 12-16 September 2005), Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 198-212.

53. Hâkon W., ***Cascading Style Sheets***, Ph.D. thesis, Faculty of Mathematics and Natural Science, University of Oslo, Norway, 2005.

54. Horn P., ***Autonomic computing: IBM's perspective on the state of information technology***. In: Online Document at http://www-128.ibm.com/developerworks/autonomic/library/ac-summary/ac-manifest.html, October 2001.

55. Kephart J. O., Chess D. M., ***The Vision of Autonomic Computing***, In: IEEE Computer, Volume 36, Issue 1, p. 41-50, January 2003.

56. Lachenal C., ***Modèle et infrastructure logicielle pour l'interaction multi-instrument multisurface***, Thèse de doctorat, 17 décembre 2004. 200 p.

57. Jabarin B., Graham N., ***Architectures for Widget-Based Plasticity***, in Proceedings of Design, Specification and Verification of Interactive Systems (DSV-IS 2003), Springer LNCS, 124-138, 2003.

58. ***Jazz/Piccolo***, http://www.cs.umd.edu/hcil/jazz/.

59. Johnson J., ***Selectors: going beyond user-interface widgets***, CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems, 1992, ISBN 0-89791-513-5, pp 273-279.

60. Johnson J.., Nardi B., Zarmer C., Miller J., ***ACE: building interactive graphical applications***, Communication ACM, volume 36, number 4, ACM Press, New York, NY, USA, 1993, pp. 40-55.

61. Kawai S., Aida H., and Saito T. ***Designing interface toolkit with dynamic selectable modality***. In Proceedings of the Second Annual ACM Conference on Assistive Technologies (Vancouver, British Columbia, Canada, April 11 - 12, 1996). Assets '96. ACM Press, New York, NY, 72-79.

62. Khushraj D., Lassila O., ***Ontological approach to generating personalized user interfaces for web services***. In International Semantic Web Conference, pp. 916-927, 2005.

63. ***LDAP***, http://www.cru.fr/ldap/.

64. Lecolinet E., ***XXL : : A Dual Approach for Building User Interfaces***, UIST'96, pp 99-108.

65. Lecolinet E., ***A Brick Construction Game Model for Creating Graphical User Interfaces: The Ubit Toolkit***. In Proc. INTERACT'99, 1999.

66. Lecolinet E. 2003. ***A molecular architecture for creating advanced GUIs***. In Proceedings of the 16th Annual ACM Symposium on User interface Software and Technology (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM Press, New York, NY, pp. 135-144.

67. Lepreux S., Vanderdonckt J., Michotte B., ***Visual Design of User Interfaces by (De)composition***, Proc. of 13th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2006 (Dublin, 26-28 July 2006), G. Doherty and A. Blandford (eds.), Lecture Notes in Computer Science, Vol. 4323, Springer-Verlag, Berlin, 2006, pp. 157-170.

68. Lepreux S., Vanderdonckt J., ***Towards Supporting User Interface Design by Composition Rules***. In Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006), Chapter 19, Springer-Verlag, Berlin, 2006, pp. 231-244.

69. Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Florins M., Trevisan D., ***UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces***, in Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" (Gallipoli, May 25, 2004), Luyten, K., M. Abrams, Limbourg, Q., Vanderdonckt, J. (Eds.), Gallipoli, 2004, pp. 55-62.

70. Limbourg Q., ***Multi-path Development of User Interfaces***, Ph.D. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 4 November 2004.

71. Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., López-Jaquero V., ***UsiXML : a language supporting multi-path development of user interface***, Lecture Notes in Computer Science, Volume 3425/2005, ISBN 978-3-540-26097-4, Springer Berlin / Heidelberg, 2005, pp. 200-220.

72. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V., ***UsiXML: a Language Supporting Multi-Path Development of User Interfaces***, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Berlin, 2005, pp. 200-220.

73. Linton M., Tang S., Churchill S. ***Redisplay in Fresco***. *The X Resource*, 1994, (9), 63-69.

74. Luyten K., Thys K., Vermeulen J. and Coninx K., ***A Generic Approach for Multi-Device User Interface Rendering with UIML***, 6th International Conference on Computer-Aided Design of User Interfaces (CADUI'2006), Bucharest, Roemania, June 5-8, 2006.

75. Martínez-Ruiz J., Muñoz Arteaga J., Vanderdonckt J., ***Transformation of XAML schema for RIA using XSLT & UsiXML***, Proc. of XlX Congreso Nacional y V Congreso Internacional de Informática y Computación de la ANIEI, Avances en Tecnologías de la Información CNCIIC'2006 (Tuxtla Gutiérrez, 25-27 October 2006), 2006.

76. ***Microsoft Speech API***, http://www.microsoft.com/speech/.

77. Myers B., Hudson S. E., and Pausch R. 2000. ***Past, present, and future of user interface software tools***. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (Mar. 2000), 3-28.

78. Nardi B. and Miller J-R. ***Twinkling lights and nested loops: Distributed prolem solving and spreadsheet development***. Int. J. Man-Machine Studies 34 (1991), pp.161-184.

79. Nogier J.F., ***Ergonomie du logiciel et design Web : le manuel des interfaces utilisateur***, 3ème édition, Dunod, 272 pages, 2005

80. ***OpenInventor***, http://oss.sgi.com/projects/inventor/.

81. ***OWL***, http://www.w3.org/2004/OWL/.

82. ***OWL-S***, http://www.w3.org/Submission/OWL-S/.

83. Paolucci M., Kawamura T., Payne TR., ***Sycara K., Semantic Matching of Web Service Capabilities***. LNCS 2342, 2002.

84. Paterno F., Mancini C., Meniconi S., ***ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models***. In Proceedings Interact'97, pp.362-369, July'97, Sydney.

85. Ponnekanti S., Lee B., Fox A., Hanrahan P., Winograd T., ***Icrafter : A service framework for ubiquitous computing environments***, In UbiComp'01, 2001.

86. Puerta, A.R. and Eisenstein, J., ***Towards a General Computa-tional Framework for Model-based Interface Development Systems***, Knowledge-Based Systems 12, 8 (1999), 433-442.

87. Prosise J., ***Programming Windows With MFC***, Second Edition, Microsoft Press, 1999, p.503.

88. Roudaut A., Coutaz J., ***Méta-IHM ou comment controler l'espace interactif ambiant***, Ubimob 2006, ACM Press.Actes des Troisèmes Journées Francophones : Mobilité et Ubiquité 2006 (Paris, France), 5-8 septembre 2006.

89. Scapin D.L., Pierret-Golbreich C., ***Towards a method for task description: MAD***, in L. Berlinguet, D. Berthelette (Eds.), Work with display units 89, 1990.

90. Sottet J.S., Calvary G., Favre J.M., ***Ingénierie de l'Interaction Homme-Machine Dirigée par les Modèles***, Premières Journées sur l'Ingénierie Dirigée par les Modèles, IDM'05, Sébastien Gérard, Jean-Marie Favre, Pierre-Alain Muller, Xavier Blanc, Editors, Paris, 30 juin-1er juillet 2005, ISBN 2-7261-1284-6, pp 67-82.

91. Sottet JS., Calvary G., Favre JM., Coutaz J., Demeure A., ***Towards Mapping and Model Transformation for Consistency of Plastic User Interfaces***, Workshop on The Many Faces of Consistency in Cross-platform Design, ACM conf. on Computer Human Interaction, CHI 2006, Montréal., 22-23 Avril 2006.

92. Sottet J.S., Ganneau V., Calvary G., Coutaz J., Demeure A., Favre J.M., Demumieux R., ***Model Driven Adaptation for Plastic User Interfaces***, In Proc. HCI International 2007, ERCIM SESAMI, Beijing, July 2007.

93. Sowa J.F, ***Conceptual Structures – Information Processing in Mind and Machine***, Addison Wesley, 1984.

94. Sowa J.F., ***Conceptual graphs for a database interface***, IBM Journal of Research and Development 20:4, 1976, pp. 336-357.

95. Sowa, J.F., ***Knowledge Representation: Logical, Philosophical, and Computational Foundations***, Brooks/Cole Publishing Co., Pacific Grove, 2000.

96. Stuerzlinger W., Chapuis O., Phillips D., Roussel. N., ***User Interface Façades: Towards Fully Adaptable User Interfaces***, In Proceedings of UIST'06, the 19th ACM Symposium on User Interface Software and Technology, pages 309-318, October 2006. ACM Press.

97. Tandler P., ***Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogenous Devices***, In Proc. of the 3rd International Conference on Ubiquitous Computing (Ubicomp 2001), Atlanta Sept. 2001, Abowd G., Brumitt B. and Shafer S. Eds., Springer, LNCS 2201, p. 96-115.

98. Thevenin D., Coutaz J., ***Plasticity of User Interfaces: Framework and Research Agenda***, In Proc. Interact99, Edinburgh, , A. Sasse & C. Johnson Eds, IFIP IOS Press Publ. pp.110-117

99. Thevenin D., ***L'adaptation en Interction Homme-Machine : le cas de la plasticité***, Thèse de doctorat Informatique préparée au Laboratoire de Communication Langagière et interaction Personne-Système (IMAG), Université Joseph Fourier.Décembre 2001. 238 p.

100. ***UDDI***, www.uddi.org.

101. Vanderdonckt J., ***Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive***, Thèse des Facultés Universitaires Notre-Dame de la Paix, Spécialité Informatique, Juillet 1997.

102. Vanderdonckt J., Berquin P., ***Towards a Very Large Model-Based Approach for User In-terface Development***, In: Paton, N.W., Griffiths, T. (eds.): Proc. of 1st IEEE Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edinburgh, September 5-6, 1999). IEEE Computer Society Press, Los Alamitos (1999) pp. 76–85.

103. Vanderdonckt J., Limbourg Q., Michotte B., Bouillon L., Trevisan D., Florins M., ***UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces***, in Proc. of W3C Workshop on Multimodal Interaction WMI'2004 (Sophia Antipolis, 19-20 July 2004).

104. Vermeulen J., Vandriessche Y., Clerckx T., Luyten K. and Coninx K., ***Service-interaction Descriptions: Augmenting Services with User Interface Models***, Engineering Interactive Systems 2007 (IFIP WG2.7/13.4 10th Conference on Engineering Human Computer Interaction, IFIP WG 13.2 1st Conference on Human Centred Software Engineering and DSVIS - 14th Conference on Design Specification and Verification of Interactive Systems), Salamanca, Spain, March 22-24, 2007.

105. ***WSDL***, www.w3.org/TR/wsdl.

106. ***XSL***, http://www.w3.org/Style/XSL/

107. ***XPath***, http://www.w3.org/TR/xpath (v1.0) et http://www.w3.org/TR/xpath20/ (v 2.0)

108. ***XSLT***, http://www.w3.org/TR/xslt (v1.0) et http://www.w3.org/TR/xslt20/ (v2.0)