Chapitre 7[1]

Architecture logicielle conceptuelle des systèmes interactifs

7.1. Introduction

L'architecture, dans son acception générale, est l'art de construire un objet selon des règles déterminées. C'est aussi la structure de l'objet construit. Ici, l'objet d'intérêt est un logiciel interactif et l'architecture est le fruit de l'activité de conception globale. Avec cette étape du processus de développement d'un logiciel, nous quittons l'espace de conception proprement dite de l'Interface Homme-Machine (IHM) pour pénétrer dans l'espace logiciel réservé aux informaticiens. Cette situation charnière [NIG 97] se traduit par une tension permanente entre les contraintes techniques imposées par les outils de mise en œuvre et la satisfaction des requis de l'utilisateur et des spécifications externes de l'IHM.

L'étude rationnelle et explicite des architectures logicielles date du début des années quatre-vingt-dix. Il s'agit donc d'un thème de recherche récent bien que tout informaticien "fasse de l'architecture" au quotidien. La jeunesse du domaine de recherche explique l'absence de consensus sur la définition de la notion d'architecture logicielle et la floraison de termes comme style, motif, architectures conceptuelle et implémentationnelle. Ce chapitre vise à clarifier la situation en présentant l'état de l'art en la matière ainsi que les points d'articulation entre les concepts qu'il convient de maîtriser. Nous traiterons des systèmes interactifs en général, les règles énoncées s'appliquant, bien entendu, au cas des systèmes

---

[1] Ce chapitre a été rédigé par Joëlle COUTAZ et Laurence NIGAY avec la participation de Nick GRAHAM.

d'information. Ici, "système d'information" ne désigne pas l'organisation du traitement au sein d'un organisme, mais le logiciel qui permet le traitement de cette information.

Ce chapitre est structuré comme suit : En première partie, nous introduisons les principes de base en matière d'architecture logicielle suivis des concepts qui lui sont fréquemment associés : style, modèle de référence et motif architectural. Nous les illustrons ensuite avec les modèles architecturaux de référence, Seeheim et Arch, puis avec les modèles multi-agent comme PAC. En dernière partie, nous décrivons le modèle hybride PAC-Amodeus et ses déclinaisons pour les interfaces multimodales et les collecticiels.

7.2. Architecture logicielle : principes de base

Les principes de base comprennent quatre volets : la définition de la notion d'architecture logicielle, le processus de conception architecturale et les concepts manipulés, enfin les systèmes de représentation et les techniques d'évaluation des structures architecturales résultant de ce processus.

7.2.1. *Définition*

L'architecture d'un système informatique est un ensemble de structures comprenant chacune : des composants, les propriétés extérieurement visibles de ces composants et les relations que ces composants entretiennent. De cette définition inspirée de [BAS 98], retenons deux points importants :

– Les propriétés *extérieurement visibles* d'un composant définissent son comportement attendu : elles traduisent les hypothèses que d'autres composants peuvent faire sur ce composant (par exemple, les services qu'il fournit, les ressources requises, ses performances, ses mécanismes de synchronisation). Le qualifiant "extérieurement visible" exprime le pendant des détails que le composant encapsule. Un *composant* est donc une *unité d'abstraction* dont la nature dépend de la structure considérée dans le processus de conception architecturale. Ce peut être un service, un module, une bibliothèque, un processus, une procédure, un objet, une application, etc. Mais le comportement observable de chaque composant fait partie de l'architecture puisqu'il détermine ses interactions possibles avec d'autres composants.

– Un système peut avoir *plusieurs structures* correspondant chacune à un point de vue, donc à une finalité ou classe de problèmes précis à résoudre. En somme, comme en architecture du bâtiment, un logiciel est représenté par plusieurs plans et schémas destinés chacun à un corps de métier et dépendants de l'étape du processus de développement.

7.2.2. *Processus de développement et concepts architecturaux*

Le processus de conception d'architecture logicielle couvre un ensemble d'activités dont l'ordre est laissé au savoir-faire du concepteur. Certaines de ces activités sont pratiquées manuellement, d'autres sont assistées par voie de spécification-génération-mécanisme d'exécution. Nous retenons les activités clefs suivantes : décompositions fonctionnelle et modulaire, allocation des fonctions aux modules, identification des processus et leurs liens avec les modules et les processeurs. Ces activités mettent en jeu des concepts architecturaux comme les modèles d'architecture de référence, la notion de style et de motifs, et conduisent à la production d'architectures conceptuelles et implémentationnelles.

La figure 7.1 illustre le processus de conception architectural depuis le modèle de référence jusqu'à l'architecture implémentationnelle.

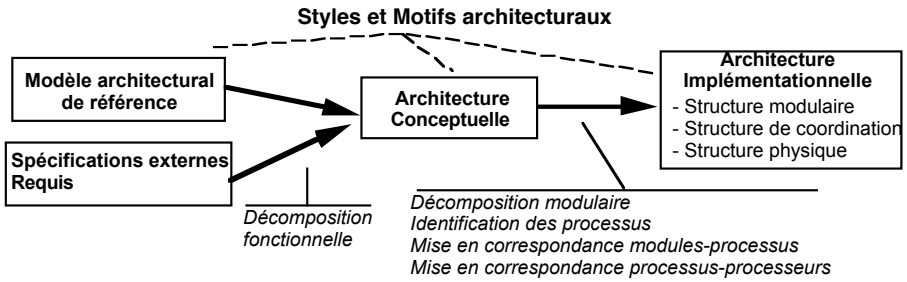

Figure 7.1. *Relations entre modèle architectural de référence, architecture conceptuelle et architecture implémentationnelle en relation avec les activités du processus de conception (décomposition fonctionnelle, décomposition modulaire, etc.). Les flèches dénotent une approche descendante du processus de conception architectural. Les traits en pointillé indiquent les notions orthogonales intervenant dans le processus de conception.*

La *décomposition fonctionnelle* du système consiste à exprimer les besoins fonctionnels du système en des unités plus simples. Le résultat de cette activité est une structure appelée *architecture conceptuelle*. Les composants sont des abstractions qui traduisent les requis fonctionnels du système. Leurs relations sont de type "échange-des-données avec". Par exemple, le schéma de la figure 7.10 est une représentation de l'architecture conceptuelle du système MATIS. L'élaboration d'une architecture conceptuelle peut s'appuyer sur un modèle architectural de référence.

Un *modèle architectural de référence* est une décomposition fonctionnelle normalisée pour un problème connu. Par exemple, un compilateur couvre quatre (ou cinq) fonctions aujourd'hui parfaitement identifiées : les analyses lexicale, syntaxique, sémantique et pragmatique, et la génération de code. En IHM, les

modèles Seeheim et PAC-Amodeus, présentés plus loin, sont des modèles architecturaux de référence. En reprenant l'exemple du système MATIS, son architecture conceptuelle est fondée sur le modèle de référence PAC-Amodeus.

La *décomposition modulaire* définit une structuration statique du système selon des règles, par exemple, de minimisation des dépendances entre les composants. Les composants sont ici des modules liés par des relations de type "est-un-sous-module-de". La structure modulaire permet au chef de projet de répartir les efforts de développement entre plusieurs équipes. En relation étroite avec la décomposition modulaire, l'*allocation des fonctions* aux modules consiste à ranger les fonctions de l'architecture conceptuelle dans des modules. Autrement dit, il s'agit de définir la couverture fonctionnelle de chaque module : un module peut regrouper plusieurs fonctions, mais une fonction peut être répartie sur plusieurs modules. Le résultat de cette activité constitue la *structure modulaire*.

L'i*dentification des processus et l'assignation de modules aux processus* consistent à décider quels modules serviront de lieu d'exécution aux processus. La *structure de coordination*, vue dynamique architecturale à l'exécution, constitue le résultat de cette activité. L'*allocation de processus* aux processeurs répartit la charge de calcul sur plusieurs calculateurs. Cette répartition peut être statique ou se reconfigurer dynamiquement. La *structure physique* du système est alors conçue. Les structures de coordination et physique complètent la structure modulaire de l'étape précédente pour constituer ensemble l'*architecture implémentationnelle*.

Le passage entre l'architecture conceptuelle et l'architecture implémentationnelle se pratique selon trois approches [AND 00] : par l'application d'heuristiques [DUV 99], par l'utilisation d'outils fondés sur un modèle architectural de référence (voir au chapitre 8 des exemples d'outils d'aide à la construction d'IHM), par des outils paramétrés ou qui permettent les retouches manuelles. L'outil TCD, fondé sur le modèle de référence Dragonfly [AND 00], permet de maintenir à tout instant la correspondance entre les architecture conceptuelles et implémentationnelles d'un système. L'expérience montre qu'au fil du temps, les modifications des spécifications externes de l'interface homme-machine sont directement injectées dans l'architecture implémentationnelle au risque de perdre la conformité avec l'architecture conceptuelle. Un outil comme TCD permet de retrouver, en situation de rétroconception, l'architecture conceptuelle à partir de l'architecture implémentationnelle.

La notion de *style d'architecture* intervient de manière orthogonale aux concepts de modèle de référence, d'architectures conceptuelle et implémentationnelle. Le choix d'un style intervient dans toutes les activités du processus. Un style d'architecture se définit selon trois volets [GAR 93] :

– Un vocabulaire d'éléments (par exemple, les concepts de filtre et de tuyau, le concept de machine abstraite),

– Des règles de configuration sur les éléments du vocabulaire (par exemple, un tuyau doit relier deux filtres).

– Une sémantique fournissant un sens à la description structurelle.

Au vu de cette définition, *un style n'est pas une architecture* mais une aide générique à l'élaboration de structures architecturales. De facto, un style véhicule des propriétés spécifiques [SHA 95]. Toute architecture repose sur un style, qu'il soit homogène ou hétérogène. En cela un style peut être compris comme une classe d'architectures. Les filtres et tuyaux (pipes and filters), les machines abstraites en couche (par exemple les modèles en "fermeture éclair" de Dewan [DEW 99] et de Patterson [PAT 94]), les organisations orientées-objets, les modèles de référence MVC [KRA 88] et PAC [COU 90], répondent à des *styles homogènes*.

Le style "taille unique" qui remplirait tous les critères est illusoire. En architecture, l'hétérogénéité est un mal nécessaire. Elle a de multiples causes :

– Elle peut survenir au cours du processus de réification. Par exemple, un modèle conceptuel de partage de données peut s'affiner à l'implémentation en mémoires locales assorties d'une communication par messages ;

– L'hétérogénéité peut apparaître au sein d'un même niveau de réification. Typiquement, les systèmes ouverts et la réutilisation rendent l'hétérogénéité incontournable. Nous verrons en 7.5 un exemple de modèle architectural de référence selon un *style hétérogène* : PAC-Amodeus [NIG 94].

La cohabitation de styles offre l'avantage de choisir le style le mieux adapté à telle ou telle entité de l'organisation structurelle. Inversement, l'hétérogénéité peut être la cause d'incompatibilités : incompatibilités entre les structures de contrôle, entre les protocoles, etc. Nous verrons le rôle des composants adaptateurs comme réponse à différentes sources d'incompatibilités.

Tout comme le style, la notion de motif intervient de manière orthogonale aux concepts de base. Un *motif* ou *pattern architectural* est une micro architecture répondant à un micro problème récurrent. On trouve des motifs architecturaux applicables aussi bien au niveau conceptuel qu'au niveau implémentationnel. Au niveau conceptuel, un motif est un micro modèle architectural de référence. Comme toute architecture, un motif répond à un style donné. On trouvera dans [GAM 95], un ensemble cohérent de motifs pour des problèmes récurrents de conception logicielle fondée sur le style à objets. Ici, la description d'un motif comprend non seulement une spécification OMT pour représenter la structure objet du motif, mais aussi des heuristiques indiquant les clauses de l'applicabilité du motif, des exemples de scénario d'utilisation, un exemple d'implémentation et une discussion sur les

difficultés et pièges de mise en œuvre. Nous verrons plus loin des exemples de motifs liés au modèle de référence PAC-Amodeus.

Notons que tout système ne justifie pas nécessairement la production systématique de plusieurs vues architecturales. Certains, parce qu'il s'agit de maquettes jetables, ne méritent pas de dossier de conception globale. Il arrive que pour des systèmes de petite taille, les structures conceptuelle et modulaire soient très proches. Si le système comprend un seul processus, la structure de coordination devient inutile. Toutefois, lorsque l'architecture d'un système doit revêtir plusieurs vues, il convient de veiller au maintien de leur cohérence au cours du processus de développement et de maintenance. En l'absence d'outil, cette tâche peut être coûteuse et source d'erreurs. Des environnements de production de logiciels sont conçus à cette fin offrant des systèmes de représentation idoines.

7.2.3. *Représentation*

Une représentation est un outil d'analyse et de communication. Comme dispositif d'analyse, une représentation permet à l'auteur de raisonner sur des alternatives de conception, d'en comprendre les implications et d'évaluer la solution au regard de critères donnés. En tant que dispositif de communication, la représentation véhicule la solution adoptée par l'auteur auprès des équipes partenaires. Il convient alors que la représentation soit non ambiguë et lisible :

– Ambiguïté : les partenaires ne doivent pas faire d'interprétations erronées sous peine de perdre la conformité des descriptions au fil du processus de réification,

– Lisibilité : l'utilisateur de la représentation doit pouvoir comprendre les conséquences de la structure sur l'exploitation qu'il va en faire. Par exemple, la décomposition modulaire doit permettre au chef de projet d'identifier la planification des livraisons et de répartir le travail entre les équipes.

Retenons qu'une représentation architecturale de système *n'est pas l'architecture* du système. Elle est *la représentation de l'une des structures de l'architecture*. Autrement dit, il est inexact de dire "ce diagramme est l'architecture globale du système", mais il est correct d'affirmer "ce diagramme représente la structure modulaire du système", ou encore, "celui-ci représente la structure de coordination du système".

Le diagramme de boîtes et de flèches est le système de représentation le plus communément employé dans les dossiers de spécifications, mais il en est bien d'autres comme l'expression de la modularité des langages de programmation et les langages d'interconnexion de composants :

– Dans les diagrammes de description architecturale, les boîtes dénotent les *composants* de la structure tandis que les flèches représentent les *connecteurs*. Les composants sont les unités de calcul essentielles de la structure alors que les connecteurs expriment leurs interactions. Les flux de données, l'appel de fonction, la communication par messages, les relations de sous-classes sont des exemples de connecteurs. Si les diagrammes offrent une lecture intuitive, boîtes et flèches ont des significations bien différentes d'une représentation à l'autre. En l'absence d'une définition précise de leur sémantique, ces diagrammes ne sont pas de bons supports aux activités d'analyse.

– Les capacités d'expression des langages de programmation, telle la notion de *package* dans Ada, rend explicites les interfaces de connexion entre composants, mais ces notations, utiles au niveau implémentationnel, sont peu lisibles pour une activité d'analyse au niveau conceptuel : L'articulation entre les composants n'est pas saillante et de plus, les composants sont supposés être de même nature. Or, la tendance s'oriente vers la construction de systèmes par composition de systèmes existants. De nombreux langages de description d'architecture motivés par cette approche sont développés à cette fin (voir dans [BAS 98, chapitre 12] une discussion approfondie sur le sujet).

Dans la suite de notre exposé, nous retiendrons les représentations par diagrammes de composants et de connecteurs. Insistons sur le fait qu'un connecteur traduit des relations entre composants pour un niveau d'analyse donné. Si l'analyse doit être poussée plus avant, ce même connecteur est à son tour représenté par un diagramme de composants et de connecteurs. Les conditions d'arrêt du processus d'affinement peuvent être : la compétence des équipes de développement (inutile de poursuivre la décomposition de ce que l'on maîtrise) ou, plus généralement, la nature des composants et mécanismes réutilisables de la plate-forme d'accueil.

Comme tout produit de conception, une solution architecturale doit faire l'objet d'une évaluation avant le codage proprement dit. En l'absence d'un contrôle précoce, les coûts de correction, on le sait, peuvent être rédhibitoires.

7.2.4. *Evaluation*

Une architecture n'est jamais intrinsèquement bonne ou mauvaise, mais sa qualité se juge à la lumière de propriétés identifiées par avance. Les requis techniques, comme la portabilité et l'efficacité, ne suffisent pas à expliquer une architecture : le contexte culturel de l'entreprise, les délais imposés de production, et de manière générale, tous les intervenants, dans le processus de conception et de mise en œuvre d'un logiciel, ont une influence sur les structures architecturales retenues. Nous retiendrons la méthode SAAM (Software Architecture Analysis

Method [KAZ 94] [BAS 98]), résolument centrée sur la mise en contexte de propriétés au moyen de scénarios représentatifs.

Un scénario SAAM doit anticiper une forme d'utilisation du logiciel et ses évolutions (rappelons ce principe de bon sens : "un bon concepteur est celui qui sait anticiper le changement"). L'éventail de scénarios nécessaires à l'analyse doit couvrir l'ensemble des représentations architecturales retenues pour le système de même que le point de vue de l'ensemble des acteurs impliqués dans le processus de développement (utilisateur final représentatif, développeur, chargé de maintenance, etc.). Par exemple, un scénario concernant à la fois l'utilisateur et le responsable en maintenance du système, traduirait la possibilité pour l'utilisateur de personnaliser l'IHM en ajoutant une barre de boutons tout en minimisant les modifications de la version originale du système à livrer dans les prochaines semaines. Une représentation structurale donnée est analysée dans le contexte concret de ces scénarios. On relèvera ici l'analogie avec les scénarios centrés sur l'utilisateur [CAR 95] et conçus pour tester les modèles de tâche ou les spécifications externes d'un système interactif.

Au-delà des propriétés usuelles du Génie Logiciel (portabilité, efficacité, modifiabilité, etc.), les systèmes interactifs sont concernés par des propriétés centrées sur l'utilisateur dont le respect peut avoir un impact sur les solutions architecturales. On consultera en annexe de ce chapitre, la liste des propriétés que nous avons élaborée comme déclinaison de la notion d'utilisabilité. Au cours de ce chapitre, nous utiliserons certaines d'entre elles comme élément d'évaluation des architectures spécifiques aux systèmes interactifs. On trouvera dans [GRA 96b] une discussion systématique des relations entre modèles d'architecture et propriétés.

Ayant introduit les principes et concepts généraux de l'architecture logicielle, nous sommes en mesure dans les sections qui suivent, de présenter des modèles de référence propres aux systèmes interactifs. Dans ce chapitre, nous indiquons comment, à partir de modèles de références, produire une architecture conceptuelle. Nous ne traiterons pas du passage de l'architecture conceptuelle à l'architecture implémentationnelle, mais on trouvera des éléments de réponse dans [PHI 99].

7.3. Modèles de référence pour les systèmes interactifs

Tous les modèles architecturaux de référence applicables aux systèmes interactifs répondent au requis de modifiabilité : en l'état des connaissances et des pratiques, la mise au point itérative d'une IHM est la seule stratégie effective. Mais le succès de cette approche repose soit sur l'existence de bons outils de construction de maquettes, soit sur la possibilité de retoucher "à la main" une IHM sans mettre en cause la fiabilité, ni atteindre des coûts prohibitifs de mise à jour. Tous les modèles

de référence répondent à cette exigence de modifiabilité avec le *principe de séparation fonctionnelle*. Un premier grain de séparation est la distinction entre les services d'une application et les fonctions chargées d'assurer l'interaction avec l'utilisateur. L'application, appelée aussi *noyau fonctionnel*, regroupe les concepts du domaine et les opérations qui leur sont applicables. L'IHM a la charge de présenter à l'utilisateur concepts et fonctions et de lui permettre de les manipuler selon un enchaînement, reflet logiciel du modèle de tâche (Cf. chapitres 3 et 4). Les modèles de référence Seeheim [PFA 85] et Arch [UIMS 92] sont des affinements de cette décomposition bipartite. Nous présentons une variante du Seeheim original puis sa version révisée : Arch.

7.3.1. *Les principes fondateurs du modèle Seeheim*

La conception itérative et sa compagne immédiate, la réutilisation logicielle, suggèrent que le noyau fonctionnel n'ait aucune connaissance des services de l'IHM et réciproquement. Ici, absence de connaissance ne signifie pas absence de communication, mais implique des références par indirection. La double nécessité d'assurer à la fois l'indépendance et les échanges d'information entre le Noyau Fonctionnel (NF) et l'IHM se traduit par l'introduction d'une unité fonctionnelle qui joue le rôle d'intermédiaire. Ainsi, et comme l'illustre la figure 7.2, un système interactif est structuré en trois unités fonctionnelles :

– Le Noyau Fonctionnel (NF) regroupe les concepts et fonctions du domaine,

– La Présentation est chargée de rendre perceptible l'état pertinent des concepts du domaine (voir la propriété d'*observabilité* en annexe) et d'en permettre la manipulation par l'utilisateur,

– Le Contrôleur de Dialogue (CD) sert de pont d'indirection entre le NF et la Présentation. En tant qu'intermédiaire, le Contrôleur de Dialogue est responsable de la gestion des relations entre le NF et la Présentation.

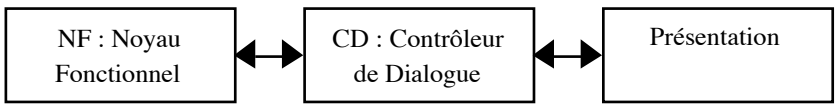

Figure 7.2. *Le modèle de référence de base pour les systèmes interactifs. Une flèche à double sens traduit l'existence d'échanges d'informations entre deux unités fonctionnelles.*

Une première classe de relations désigne la correspondance entre les concepts du domaine et les interacteurs constitutifs de la Présentation. Nous rappelons qu'à tout concept du domaine doit, selon la Théorie de Norman, correspondre au moins un interacteur. Un *interacteur* est un instrument logiciel, entité capable de restituer une

abstraction auprès de l'utilisateur sur un dispositif de sortie (écran, haut-parleur, etc.) et de lui permettre de manipuler cette abstraction via des dispositifs d'entrée (clavier, souris, microphone, doigt, etc.). Le lien entre les concepts du domaine et leurs interacteurs, c.-à-d. le mécanisme d'indirection, est géré par le CD. Une seconde classe de relations concerne l'enchaînement des appels de service entre le NF et la Présentation. L'enchaînement des appels doit être, en principe, conforme aux relations temporelles et structurelles du modèle de tâche produit très en amont dans le processus de conception du système interactif. Cette conformité est caractérisée par les propriétés d'*atteignabilité* et d'*interaction multifilaire* (cf. annexe). Le NF et la Présentation définissent deux perspectives d'un même domaine, mais chaque perspective a un rôle précis à remplir. Dans le NF, la modélisation du domaine est guidée par des considérations de calcul alors que dans la Présentation, le modèle est conçu pour manipuler et rendre perceptible l'état du NF. En conséquence, le NF et la Présentation utilisent des formalismes et des techniques de représentation différents. Le pont entre les deux mondes est assuré par le CD au moyen de deux protocoles de communication, l'un adapté aux besoins du NF, l'autre à ceux de la Présentation.

7.3.2. *Le modèle de référence Arch ou Seeheim révisé*

Comme le montre la figure 7.3, le modèle Arch [UIMS 92] s'appuie sur les composants fonctionnels de Seeheim. On y retrouve le Noyau Fonctionnel (NF), le Contrôleur de dialogue et la Présentation. Les pieds de l'arche constituent les composants imposés par la réalité : le NF réalise les concepts du domaine ; le Composant d'Interaction, en contact direct avec l'utilisateur, est mis en œuvre au moyen des interacteurs (*widgets*) d'une boîte à outils. En clé de voûte, le Contrôleur de Dialogue gère l'enchaînement des tâches ainsi que les liens entre les objets regroupés dans ses deux composants voisins : l'Adaptateur de Domaine et la Présentation. L'Adaptateur de Domaine sert, pour l'essentiel, à ajuster les différences de modélisation des objets conceptuels entre le NF et le Contrôleur de Dialogue. Le composant de Présentation joue un rôle similaire : il permet au Contrôleur de Dialogue de s'affranchir du fonctionnement de la boîte à outils du niveau interaction. La Présentation peut se voir comme une boîte à outils virtuelle qui implémente des objets de présentation concrétisés en fin de compte par les interacteurs d'une boîte à outil réelle.

La figure 7.3 doit se voir comme une représentation canonique. En réalité, l'importance relative des cinq composants de l'arche varie en fonction du cas à traiter. Cette migration des fonctions entre les composants s'exprime métaphoriquement au moyen du métamodèle *slinky*, du nom de ce jouet qui, une fois mis en mouvement, voit sa masse se déplacer dynamiquement. Le choix de la répartition des fonctions entre les composants relève du savoir-faire en ingénierie du logiciel. L'équilibre doit être le résultat d'un dosage analytique des facteurs qualité

comme l'efficacité et la portabilité. On appellera *instance d'Arch*, un exemplaire de modèle Arch produit en appliquant *slinky* à un système interactif particulier.

Seeheim et Arch fournissent des structures fonctionnelles canoniques à gros grain. Ces modèles sont utiles comme cadres structurants pour une conception ou une analyse grossière de la décomposition fonctionnelle d'un système interactif. Les modèles de référence multi-agent visent une décomposition fonctionnelle plus fine avec, en filigrane, le support du parallélisme.

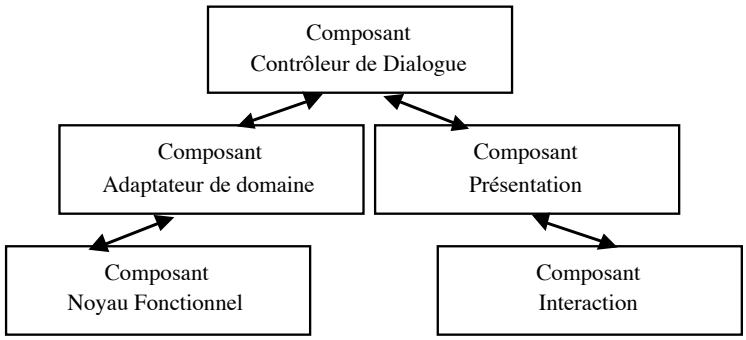

Figure 7.3. *Les composants du modèle Arch*

7.4. Modèles multi-agent

De nombreux modèles relèvent de l'approche multi-agent. Dans ce qui suit, nous en donnons les principes généraux et développons notre propre travail sur le sujet avec le modèle PAC. Nous concluons cette section par une brève analyse comparative de PAC avec d'autres modèles tout aussi pertinents.

7.4.1. *Le concept d'agent et architecture multi-agent*

Un agent est un système de traitement de l'information : il se distingue par un jeu d'opérations, des mécanismes d'entrée/sortie et une capacité à représenter un état que l'on appelle vecteur d'état. Un agent est un acteur communicant : il assume un rôle (c'est un acteur) et se manifeste par un comportement observable via l'acquisition et la production d'informations (il communique). L'acquisition et la production d'information sont des actions dont la réalisation se traduit par des événements. Les informations proviennent ou sont destinées à d'autres agents : l'agent vit en communauté. Il conduit ses activités en parallèle avec celles de ses

confrères. Il faut l'opposer à milieu d'activités séquentielles et à entité passive et isolée.

En Intelligence Artificielle Distribuée, on convient de distinguer les agents réactifs des agents intelligents [FER 95]. Les premiers, du type stimuli-réponse, ont un comportement câblé. Les seconds, dotés de mécanisme de planification, poursuivent des buts de manière adaptative. En Interface Homme-Machine, on parle implicitement d'agents réactifs. Par exemple, dans le modèle des dispositifs d'entrée de Mackinlay et de ses coauteurs [MAC 90], un dispositif est un agent réactif. Les interacteurs décrits dans [FAC 93], ou les agents PAC sont également des agents réactifs. C'est pourquoi on leur préfère souvent le terme d'*interacteur*. Un système multi-agent est une société d'agents dont la nature et l'organisation définissent une structure du système. Le modèle PAC définit une telle structure.

7.4.2. *Agent PAC : un agent à facettes*

Un agent PAC, constituant de système interactif, est un système interactif. D'après le modèle Seeheim, un système interactif peut s'analyser selon trois facettes fonctionnelles : le noyau fonctionnel, le contrôleur de dialogue, et la présentation. Un agent PAC se modélise selon ces trois perspectives complémentaires : la Présentation, l'Abstraction et le Contrôle.

– La *Présentation* définit le comportement perceptible de l'agent auprès d'un agent humain. Elle concerne à la fois les entrées et les sorties, c'est-à-dire les modalités d'action accessibles à l'utilisateur et la restitution perceptible. La Présentation : (1) interprète les événements résultants des actions physiques que l'utilisateur applique à l'agent via des dispositifs d'entrée et (2) engendre des événements qui traduisent des actions physiques du système sur les dispositifs de sortie.

– L'*Abstraction*, avec ses fonctions et ses attributs internes, définit la compétence de l'agent indépendamment des considérations de présentation. Elle constitue, au sens de Seeheim, le noyau fonctionnel de l'agent.

– Le *Contrôle* a un double rôle : (1) il sert de pont entre les facettes Présentation et Abstraction de l'agent, (2) il gère des relations avec d'autres agents PAC. Tout échange d'informations entre l'Abstraction et la Présentation s'effectue via le Contrôle. C'est aussi par leurs parties Contrôle que deux agents PAC communiquent.

Le rôle d'intermédiaire dans la communication attribue au Contrôle les fonctions d'arbitrage et de traduction :

– L'arbitrage recouvre la synchronisation et la coordination. Ces services sont nécessaires entre agents mais aussi entre l'Abstraction et la Présentation d'un agent.

En effet, ces dernières peuvent, dans certaines circonstances, être le lieu d'activités asynchrones qui requièrent alors une coordination.

– La traduction concerne la transformation de formalismes entre l'Abstraction et la Présentation qui, en raison de rôles distincts, utilisent des systèmes de représentation différents.

À titre illustratif, prenons l'exemple du thermomètre pour représenter la notion de température dans un système de contrôle industriel. Le thermomètre est l'agent PAC de la figure 7.4 a). La Présentation de cet agent sait dessiner une forme de thermomètre et reçoit les événements qui traduisent les actions de l'utilisateur ; l'Abstraction, avec son vecteur d'état "température initiale, température finale, et variation de température", constitue un modèle abstrait de l'état du thermomètre. Le Contrôle effectue le pont entre les deux facettes fonctionnelles en gérant la correspondance des phénomènes abstraits et concrets. Par exemple, l'action utilisateur qui consiste à déplacer la barre de mercure pour spécifier la température souhaitée se manifeste à la Présentation par un événement de type "localisation (x, y)". La Présentation en déduit une nouvelle hauteur de mercure qu'elle transmet au Contrôle selon un protocole convenu. Par exemple, la Présentation pourrait transmettre une valeur liée au formalisme graphique, et notamment une longueur de segment de droite. Cette longueur n'est pas nécessairement la longueur effective de la droite qui représente le niveau de mercure sur l'écran, mais une valeur correspondant à une taille de thermomètre normalisée : cette valeur serait la même quelle que soit la taille de l'exemplaire de thermomètre effectivement utilisé, en supposant par exemple que l'image du thermomètre soit proportionnelle à la taille de la fenêtre support.

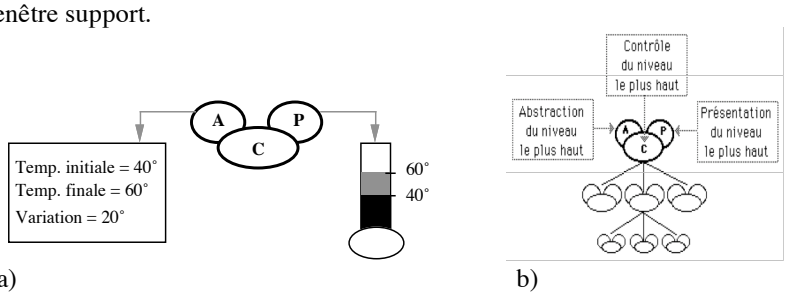

Figure 7.4. *a) Un exemple d'agent PAC. b) Le modèle d'architecture PAC. Les traits pleins représentent des échanges entre agents de niveaux d'abstraction distincts.*

Le Contrôle de l'agent thermomètre traduit la longueur du segment de droite en fonction du protocole convenu avec l'Abstraction. Ici, on échangera une valeur de température normalisée (éventuellement indépendante du fait qu'il s'agit d'une représentation en Celsius ou en Fahrenheit). L'Abstraction en déduit une température finale, calcule la variation et transmet le ou les résultats pertinents au

Contrôle (par exemple, la variation et la température finale). Ce dernier exprime les résultats dans le formalisme attendu de la Présentation qui peut alors effectuer une mise à jour du rendu perceptible de l'agent : un mercure en grisé pour dénoter la variation et la valeur en Celsius de la température souhaitée. On peut imaginer que le flux d'échanges ci-dessus soit déclenché dès la réception, par la Présentation, d'un événement "mouse-down" et qu'il soit entretenu jusqu'au relâchement du bouton de la souris. Une fois le bouton relâché, l'agent thermomètre et l'utilisateur mettent fin à la transaction, mais l'agent thermomètre se doit de signaler à d'autres agents du système le souhait final de l'utilisateur. Une autre possibilité est que le flux d'échanges ci-dessus soit déclenché seulement lorsque l'utilisateur relâche le bouton de la souris. Entre les actions enfoncement et relâchement, la Présentation prend à sa charge un retour d'information de nature purement lexicale. Par exemple, la hauteur du mercure suit le mouvement de la souris mais sans impression de la valeur correspondante de la température. Dans le premier cas, le retour d'information pendant la transaction fait intervenir les compétences sémantiques de l'agent. Dans le second cas, il ne s'agit que de feed-back lexical.

De manière générale, un événement utilisateur peut avoir un effet sémantique ou non. Un événement sans effet sémantique est traité en local par la Présentation qui produit un retour d'information perceptible. Les événements à effet de bord sémantique sont non seulement traités localement par la Présentation pour les retours d'informations lexicaux, mais, en plus, sont transmis à l'Abstraction par l'intermédiaire du Contrôle pour complément de traitements. L'enrichissement sémantique du retour d'information peut se poursuivre lorsque l'agent fait appel à d'autres agents de la hiérarchie y compris le noyau fonctionnel.

La description que nous venons de donner d'un agent PAC est applicable à tous les niveaux d'affinement ou d'encapsulation d'un système : PAC est un modèle récursif.

7.4.3. *PAC : un modèle récursif*

Comme le montre la figure 7.4 b), PAC [COU 90] structure récursivement un système interactif sous forme d'une hiérarchie d'agents. La hiérarchie traduit des relations entre agents et reflète un continuum de niveaux d'abstraction depuis le noyau fonctionnel jusqu'aux éléments fins de l'interaction, c'est-à-dire les actions physiques sur les dispositifs d'entrée/sortie. De manière générale, les agents PAC d'un système sont les entités qui réalisent les fonctions d'abstraction et de concrétisation entre les représentations conceptuelles abstraites maintenues dans le noyau fonctionnel et les phénomènes concrets de l'interaction physique. *Une hiérarchie d'agents PAC ne représente pas des relations entre classes et sous-classes d'objets* : PAC n'est pas une architecture implémentationnelle. Notons

cependant que les liens entre les agents ne sont pas nécessairement des canaux de communication. Ils peuvent traduire aussi des relations de composition c'est-à-dire des opérations de conception logicielle que sont l'encapsulation et l'affinement. Pour illustrer cette double interprétation du modèle PAC, analysons la racine de la hiérarchie.

"L'abstraction du niveau le plus haut" correspond au noyau fonctionnel du modèle de Seeheim. Le "contrôle du niveau le plus haut" peut s'interpréter de deux façons distinctes : (a) comme le contrôleur de dialogue de Seeheim ou (b) comme un agent simple dont les unités de traitement sont des unités informationnelles, c'est-à-dire un système de représentation des concepts qui fait sens pour le noyau fonctionnel. Dans le premier cas, le sous-arbre d'agents représente un affinement du contrôleur. La liaison entre l'agent du niveau le plus haut et le sous-arbre est une relation de composition (encapsulation/affinement). Dans le second cas, le sous-arbre désigne les agents dont l'agent racine a la charge. La liaison entre l'agent du niveau le plus haut et le sous-arbre est une relation d'échange d'information. De même, la "présentation du niveau le plus haut" peut se voir comme le composant Présentation du modèle Seeheim. Cette interprétation, semblable au cas (a) de la composante contrôle, correspond à l'encapsulation de la présentation du système. Dans ce cas, la présentation du niveau le plus haut se définit comme l'ensemble des présentations des agents du sous-arbre. Dans l'interprétation (b), elle est la facette de concrétisation de la racine considérée comme agent simple.

Ces deux formes d'interprétation, encapsulation/affinement et communication, sont applicables ensemble à tout agent de la hiérarchie. Dans la pratique, la vision encapsulation/affinement définit le niveau de détail auquel le concepteur logiciel s'intéresse. Par exemple, Seeheim est une architecture PAC super encapsulante. Un concepteur logiciel peut s'en contenter pour des cas de figure simples où il n'existe pas de dialogue à plusieurs fils et pour lesquels les interacteurs (widgets) d'une boîte à outils conviennent à la restitution des concepts du noyau fonctionnel. La vision communication met en évidence les liens de dépendance entre les futurs composants logiciels.

7.4.4. *Analyse critique de PAC*

PAC fournit un cadre de construction systématique applicable à tous les niveaux d'abstraction d'un système interactif. Cette approche récursive permet de concevoir une architecture progressivement de façon ascendante ou descendante. PAC permet de traduire à la fois l'encapsulation/l'affinement et la dépendance fonctionnelle, deux activités de conception logicielle intimement reliées. Notons que nous retrouvons cette double propriété dans le modèle des dispositifs d'entrée de Mackinlay [MAC 90] avec les opérateurs de fusion et de connexion. PAC permet de

distinguer les services abstraits des techniques d'interaction en introduisant un intermédiaire explicite : le Contrôle. Cette propriété d'indépendance présente plusieurs avantages :

– La satisfaction du critère qualité de réutilisabilité des constituants de l'interface et ceci de manière systématique à tous les niveaux d'abstraction.

– La modification de l'interface à moindre coût. Notre expérience avec PAC indique qu'il est possible de mettre au point une interface de façon itérative et de la faire évoluer facilement.

PAC encourage la répartition des traitements sémantiques et syntaxiques. Cette propriété présente plusieurs retombées intéressantes :

– Les constituants de l'interface communiquent au niveau d'abstraction voulu. En particulier, le noyau fonctionnel a la possibilité de s'exprimer dans son formalisme, à son niveau d'abstraction, ce qui augmente son indépendance vis-à-vis des constituants perceptibles de l'interface.

– Une part de la connaissance du noyau fonctionnel peut être déportée dans l'interface. Cette forme de délégation combine performance et qualité sémantique des retours d'information sans mettre en cause la distinction entre le noyau fonctionnel et l'interface. Dans l'exemple du thermomètre de la figure 7.4 a), le retour d'information aurait pu faire appel aux compétences du noyau fonctionnel pour décider du bien fondé de l'augmentation de température. La vérification de cette contrainte, pour des raisons de performance, aurait pu être déportée dans la partie Abstraction de l'agent thermomètre. La décision de cette migration peut être dynamique (le système en prend la décision, suite à une analyse de charge) ou bien statique décidée *a priori* par le concepteur logiciel. Nous reviendrons sur la délégation sémantique à propos de PAC-Amodeus.

Ce qui vient d'être énoncé pour PAC vaut en général pour d'autres modèles multi-agent.

7.4.5. *Modèles multi-agent et style architectural*

Tous les modèles multi-agent pour système interactif répondent à des préoccupations communes : modularité, distinction explicite entre présentation et abstraction, encapsulation, parallélisme. Tous définissent un vocabulaire d'éléments conceptuels (par exemple, pour PAC, le concept d'agent et ses facettes fonctionnelles P, A, C, et la communication par événement) ; tous imposent des contraintes entre ces éléments conceptuels (dans PAC, les agents ne communiquent que via leur facette C ; de même, les facettes P et A d'un agent ne communiquent que par la facette C de l'agent) ; tous ont une sémantique – certes plus ou moins

formelle : ce sont des styles. Ou, pour reprendre une expression courante, chaque modèle multi-agent *a son style*. Voyons comment.

Dans MVC [KRA 88], ALV [HIL 92], GIO [FAC 93] et CLOCK [GRA 96a], les agents sont, comme dans PAC, structurés en facettes fonctionnelles. Le "Model", qui définit la compétence de l'agent MVC, correspond à la composante Abstraction de PAC et d'ALV. La "View" recouvre la fonction de restitution de l'agent et le "Controller", sa fonction d'acquisition des actions utilisateur. Le couple "View, Controller" est équivalent à la facette Présentation de PAC et à la facette "View" d'ALV. AMF-C, affinement fonctionnel de PAC, est enrichi de facettes fonctionnelles aux interfaces bien formalisées [TAR 97] [TAR 98]. De même, les objets de DIANE^{++} comblent les lacunes formelles de PAC [TAR 93].

GIO est inspiré de la pratique en graphique. Comme le montre la figure 7.5, un agent GIO est une unité de calcul qui définit un niveau d'abstraction. La facette "Measure" acquiert des informations de l'agent GIO plus bas dans la hiérarchie des niveaux. La facette Abstraction délivre une information enrichie à un agent plus haut dans la hiérarchie. Le même processus est appliqué entre les facettes "Collection" et "Restitution". Cette organisation des agents en couches abstraites relève du style adopté par C2 [TAY 96].

L'agent GIO se veut réutilisable. En conséquence, les contraintes propres à un assemblage particulier d'agents sont exprimées via des agents externes au modèle appelés "Control". On retrouve ici le concept d'agent PAC, mais sans les facettes A et P. Alors que PAC et ALV explicitent les liens entre les perspectives Abstraction et Présentation d'un agent via une facette spécialisée (le "Contrôle" de PAC et le "Link" d'ALV), GIO et MVC laissent ce problème en suspens. De fait, il est fréquent de voir des amendements pratiques de MVC en M2VC dans lesquels un "Model" sert d'intermédiaire explicite entre les trois autres facettes.

Considérons en quelques lignes le niveau implémentationnel des modèles multi-agents :

– Les facettes fonctionnelles d'un agent ne sont pas nécessairement regroupées en un composant. Par exemple, dans MVC, un agent est réalisé par une instance de Modèle, une instance de Controller et une instance de View reliées par des appels de méthodes. Chaque instance de Modèle (Controller, View) est une classe d'une arborescence de classes de Modèle (Controller, View). Il en est de même pour les A et V de ALV. Inversement, l'implémentation d'un agent PAC en langage C peut se faire sous forme d'un seul module. P.Os.T. (Présentation, Objet de Simulation, Traducteur) [MOS 94], un modèle implémentationnel de PAC adapté aux systèmes interactifs de type contrôle de procédé, les facettes P, Os (qui est le A de PAC), et T (qui est le C de PAC réduit à l'expression des dépendances entre le P et le A d'un agent), sont mis en œuvre par des objets distincts.

– Les liens de dépendances exprimés au niveau conceptuel par une facette fonctionnelle n'est pas nécessairement implémentée par un composant mais par un connecteur. Par exemple, dans une implémentation de PAC avec le langage d'expression des dépendances comme FLO [DER 94], la facette "Contrôle" serait un connecteur. Il en est de même avec Rendez-vous [HIL 94] qui implémente le "L" de ALV par un connecteur. CLOCK, dont les facettes fonctionnelles d'agent reprennent celles de MVC, traduit la facette fonctionnelle C de PAC par un connecteur.

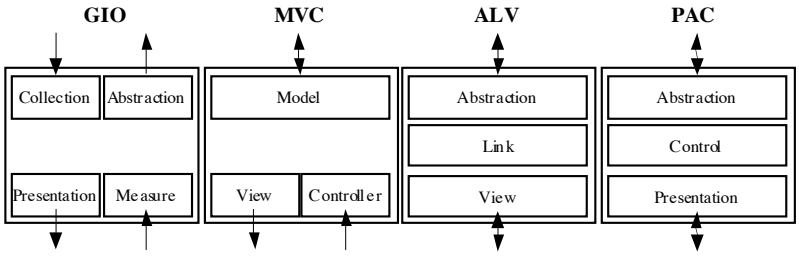

Figure 7.5. *Les facettes d'agents dans les modèles GIO, MVC, ALV, PAC*

Les modèles multi-agent relèvent de styles homogènes. Un style homogène convient si aucun composant logiciel n'est réutilisé comme c'était le cas avant l'apparition des boîtes à outils et des générateurs d'interfaces. A contrario, si des composants logiciels sont réutilisés dans le système, que ce soit des outils de développement d'IHM ou un noyau fonctionnel à rendre interactif [DUV 99], il convient alors de transformer ces composants afin de respecter le style adopté. Cette transformation ne se fait pas à moindre coût. Les modèles hybrides répondent à la contrainte d'hétérogénéité. Avant de présenter ci-dessous nos propres travaux en la matière avec PAC-Amodeus, notons un autre modèle hybride H4 [GUI 95] : H4 reprend le découpage fonctionnel de Arch tout en autorisant des communications directes entre les deux adaptateurs (Adaptateur de Domaine et Présentation) et structure quatre composants du modèle Arch en hiérarchies d'interacteurs. Nous verrons au paragraphe suivant qu'H4 présente de nombreux points communs avec PAC-Amodeus.

7.5. PAC-Amodeus : un modèle hybride

Après une brève présentation des principes de PAC-Amodeus [NIG 94], nous introduisons ses heuristiques d'application que nous illustrons avec MATIS, un système de renseignements sur les horaires d'avion. Puis nous déclinons PAC-Amodeus au cas des interfaces multimodales et des collecticiels.

7.5.1. *Définition et principes*

Comme le montre la figure 7.6, PAC-Amodeus reprend le découpage fonctionnel de Arch et structure le contrôleur de dialogue en agents PAC. Ces choix se justifient ainsi :

– Le découpage met en valeur les fonctions essentielles d'un logiciel d'interaction et notamment le contrôleur de dialogue, point sensible de la conception logicielle. Il permet également, via les deux adaptateurs, de définir des points de résolution en cas d'incompatibilité de styles, et des lieux de réutilisation de code (par exemple, les services des boîtes à outils).

– La structuration du contrôleur de dialogue en agents PAC rend explicites les interactions à plusieurs fils, fournit le bon niveau de granularité en vue de la mise en œuvre, permet via les facettes A de jouer sur l'allocation et la migration des services du noyau fonctionnel dans l'IHM.

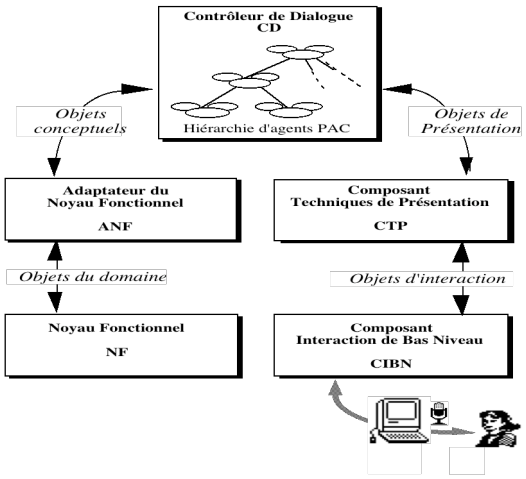

Figure 7.6. *Le modèle hybride PAC-Amodeus.*

Comme dans Arch, l'*Adaptateur de Noyau Fonctionnel* (ANF) de PAC-Amodeus sert à absorber les changements entre ses voisins directs. C'est le composant approprié pour la mise en œuvre des améliorations et de la délégation sémantiques :

– *Améliorations sémantiques* : en raison d'une erreur d'analyse ou par contraintes logicielles et matérielles, un concept peut être modélisé dans le NF d'une manière inadaptée aux besoins de l'utilisateur. Par exemple, ce concept est découpé en de multiples structures disjointes alors qu'il constitue une unité cognitive. Une amélioration (ou une réparation sémantique) peut être pratiquée dans l'ANF en

regroupant plusieurs entités du NF au sein d'un objet conceptuel ou inversement, par découpage d'entités en plusieurs objets conceptuels.

– La *délégation sémantique* relève de l'ingénierie logicielle. Elle consiste à déporter dans l'IHM, et notamment dans l'ANF, les compétences du Noyau Fonctionnel en vue d'améliorer les performances. La qualité sémantique des retours d'information exige de fréquents allers et retours entre l'IHM et le noyau fonctionnel. Lorsque la chaîne de transformations et de transferts entre l'utilisateur et le NF ne présente pas les performances requises (c.-à-d. n'est pas en accord avec les temps de réponse attendus par l'utilisateur), la décharge dans l'IHM (par exemple, sous formes de caches) doit être pratiquée. L'ANF constitue l'un de ces lieux de délégation lorsque le noyau fonctionnel est exécuté à distance. Mais la délégation peut se pratiquer dans tous les composants de l'IHM.

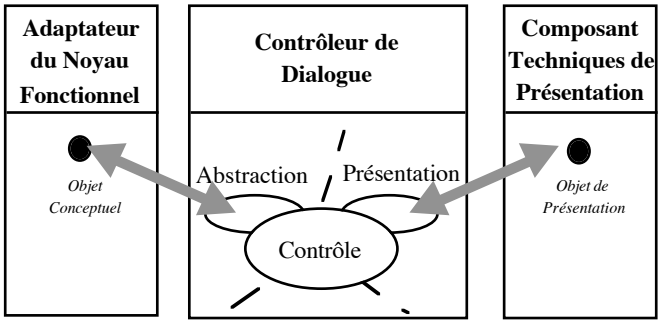

Figure 7.7. *Un agent PAC du Contrôleur de Dialogue. Le style PAC-Amodeus impose les contraintes de communication suivantes : un agent communique verticalement avec d'autres agents via sa facette C (traits en pointillé). Les facettes A et P d'un agent communiquent entre elles via la facette C de l'agent (comme dans PAC). Via sa facette A, un agent communique avec un objet conceptuel de l'ANF (ou du NF, si l'ANF n'existe pas). Symétriquement, il communique avec un objet de présentation via sa facette P. Comme dans PAC, la hiérarchie d'agents du CD n'est pas une relation de type "est-sous-classe-de".*

L'ANF échange des objets conceptuels avec son voisin IHM immédiat, clé de voûte de l'architecture : Le *Contrôleur de Dialogue* (CD). Au premier niveau de résolution, le CD peut être perçu comme le monolithe de Arch. Par affinements successifs, le concepteur logiciel identifie les niveaux d'agents nécessaires aux opérations d'abstraction/concrétisation selon les règles présentées dans la section suivante. La facette A d'un agent modélise sa compétence opératoire tout comme le Noyau Fonctionnel représente le fonctionnement conceptuel du système. Comme pour l'ANF, cette facette peut être exploitée pour y pratiquer des délégation et amélioration sémantiques. Comme le montre la figure 7.7, elle est généralement reliée à des objets conceptuels de l'ANF. La facette présentation d'un agent définit son rendu vis-à-vis de l'utilisateur. Elle est reliée à un (ou plusieurs) objets de

présentation et peut, si besoin est, maintenir des informations d'état en vue de contrôler son comportement perçu par l'utilisateur. Comme dans PAC, la facette contrôle d'un agent est un transformateur de formalisme entre les deux facettes horizontales qu'il relie et intervient dans la chaîne verticale des processus d'abstraction et de concrétisation.

L'illustration de la figure 7.7 est une vue conceptuelle. En réalité, un agent peut ne pas être relié à un objet conceptuel. Alors, il ne dépend pas du noyau fonctionnel, mais participe par exemple à une réparation sémantique. Un agent a toujours une facette contrôle (il doit communiquer avec ses pairs) mais ses facettes Abstraction et Présentation peuvent être absentes. La facette A est absente lorsque l'agent est en correspondance directe avec un objet conceptuel. Afin d'éviter des duplications d'information, la compétence abstraite de l'agent est reléguée dans l'objet conceptuel et l'abstraction se limite à l'identification de l'objet conceptuel. La facette Présentation est absente lorsque l'agent est utilisé comme unité de calcul ou de contrôle. De tels agents ne peuvent alors être des feuilles de la hiérarchie. Un agent feuille a toujours une Présentation.

7.5.2. *Heuristiques et motifs architecturaux*

Les motifs PAC-Amodeus concernent la décomposition modulaire du Contrôleur de Dialogue en agents PAC. L'application des heuristiques et motifs PAC-Amodeus exigent la connaissance des spécifications externes du système. On trouvera dans [NIG 94] le détail des justifications de ces règles et dans [NIG 97], un exemple complet de leur application. On en reprend ici les traits essentiels.

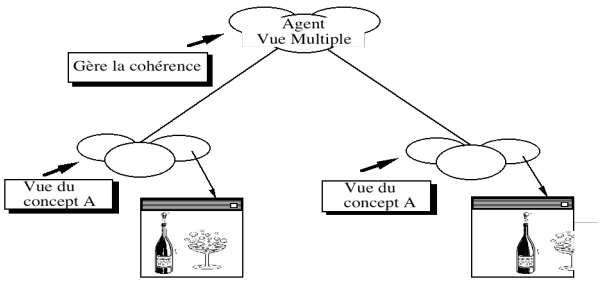

Figure 7.8. *Motif "Vue multiple d'un même concept". Un agent Père gère la cohérence.*

Règle 1 : *Une fenêtre qui sert de support à un espace de travail est modélisée par un agent.* La Présentation d'un agent "espace de travail" gère la fenêtre en tant que telle au moyen des services du système de fenêtrage : déplacement, changement

de taille, mise sous forme d'icônes, etc. Si cet agent est une feuille de hiérarchie PAC : sa Présentation a de plus la charge de restituer les concepts contenus dans l'espace de travail, son Abstraction maintient une représentation abstraite des concepts restitués, par exemple les liens logiques entre ces concepts.

Règle 2 : *Les vues multiples d'un même concept sont gérées par un agent "vue multiple*" chargé de maintenir la cohérence entre les vues (cf. en annexe, la propriété multiplicité des vues). Si l'utilisateur ou le NF agit sur l'une des vues du concept, la modification doit être répercutée sur les autres. Si la répercussion ne doit pas avoir lieu, il faut aussi être capable de le décider. Ces fonctions reviennent à l'agent vue multiple. La figure 7.8 représente le motif "vue multiple".

Règle 3 : *Une palette de classes de concepts et une barre de menus sont modélisées par un agent.* Il est fréquent que les classes de concepts du noyau fonctionnel soient regroupées dans une palette (Par exemple, la liste des figures géométriques d'un éditeur de dessin). Il est fréquent que ces palettes aient à mémoriser un "mode", par exemple le mode "cercle" dans MacDraw et que ce mode nécessite d'être répercuté. Dans ce cas, et pour augmenter la réutilisabilité de ce type de comportement, nous recommandons de modéliser une palette de concepts par un agent. L'Abstraction d'un agent palette contient la liste des classes de concept et maintient l'identité de la classe sélectionnée (le mode). La Présentation d'un agent palette est réalisée, selon le cas, dans les termes du composant Techniques de Présentation ou Techniques d'Interaction. Par exemple, si l'on dispose de Motif, la Présentation de la palette fait référence à un widget de classe "rowcolumn" dont les conditions "callback" sont reliées à des procédures de la facette Présentation. Le widget s'occupe des retours d'information lexicaux (par exemple la mise à jour des éléments en reverse vidéo de la palette). Les retours "sémantiquement plus riches" sont activés via les procédures callback. La facette Présentation peut effectuer des compléments de feedback puis transmettre le résultat de la sélection au Contrôleur de l'agent. Si la Présentation n'a pas lieu de compléter le "feedback" câblé du widget, les procédures callback peuvent alors être directement attachées au Contrôle de l'agent. Le Contrôle de l'agent palette traduit l'action de sélection en une information sémantiquement plus riche (opération d'abstraction). Dans le cas de MacDraw, il s'agirait de traduire la sélection du ième élément du "rowcolumn" en la spécification du mode de dessin. Ensuite, il avertit son environnement (en général, l'agent père) de la sélection d'une nouvelle classe de concept. Un raisonnement similaire est applicable aux barres de menus qui gèrent les menus jaillissants dans sa partie Présentation. Un sous-menu détachable, qui prend le statut de palette de concepts, devient, lorsqu'il est détaché, un exemplaire d'agent dont le père est identique à l'agent barre de menus.

Règle 4 : *Une zone d'édition est modélisée par un agent.* Si une fenêtre contient une zone d'édition, zone où l'utilisateur peut créer et modifier des concepts, celle-ci doit être modélisée par un agent. (S'il s'agit d'une simple surface d'affichage, elle est alors assurée par l'agent "espace de travail" englobant.). L'agent "zone d'édition" gère les manipulations possibles sur les concepts qu'il présente ainsi que les éventuelles contraintes applicables à ces concepts. Sa compétence consiste à gérer son contenu en tant qu'ensemble de concepts. A leur tour, les concepts sont modélisés par des agents distincts, fils de l'agent "espace de travail", sauf si ceux-ci sont atomiques (se référer à la règle suivante). Les contraintes entre les concepts fils peuvent être de nature géométrique et propres à l'agent de restitution. Il s'agit alors de contraintes lexicales. D'autres peuvent être de nature sémantique et se traduire ensuite en termes de contraintes lexicales. Si la gestion des contraintes sémantiques est à la charge de l'agent, on assiste alors à une délégation sémantique.

Règle 5 : *Un concept complexe d'une zone d'édition est modélisé par un agent.* Nous dirons qu'un concept est complexe s'il remplit l'une au moins des conditions suivantes : il est composé ou bien il est atomique mais sa présentation est composée.

– Un concept est composé lorsqu'il est constitué de sous-concepts et que cette structuration doit être reflétée à l'interface dans une même fenêtre. En général une hiérarchie d'agents isomorphique à la structure du concept composé convient à la restitution du concept composé.

– Une Présentation composée résulte de la fusion de plusieurs objets de présentation du niveau Technique de Présentation (ou du niveau Techniques d'Interaction) mais cette fusion ne peut être modélisée comme un objet de présentation (ou d'interaction). Dans ces conditions, l'agrégation doit être réalisée par la facette Présentation d'un agent.

Règle 6 : Si, depuis une fenêtre "espace de travail" contenant un concept de classe donnée, il est possible d'ouvrir un espace de travail sur un autre exemplaire de la même classe, ces deux espaces sont modélisés comme des agents fils d'un agent père commun. Il peut arriver que depuis un espace de travail qui contient un concept de classe donnée, on ouvre un nouvel espace sur un autre exemplaire de cette classe. Par exemple, plusieurs zones d'édition peuvent être ouvertes sur des documents distincts, chaque zone disposant d'un bouton d'appel à l'opérateur "ouvrir". Dans ce cas, il convient de relier les agents d'édition sous le contrôle d'un père commun qui peut, par exemple, maintenir des variables d'état communes aux zones. La vue multiple d'un même concept est un cas particulier de cette règle puisque, au lieu de l'ouverture de plusieurs exemplaires de la même classe, il s'agit de l'ouverture multiple du même exemplaire de concept.

Règle 7 : Si, depuis une fenêtre d'édition qui restitue, de manière synthétique, un ou plusieurs concepts composés, il est possible d'ouvrir une nouvelle fenêtre représentant plus en détail le contenu d'un de ces concepts composés, l'agent qui modélise la nouvelle fenêtre est fils de l'agent source. Les objets d'interaction des boîtes à outils incitent les concepteurs à restituer les concepts composés hiérarchiques sous forme de fenêtres que l'utilisateur ouvre en cascade. Par exemple, le contenu d'un répertoire est dévoilé par l'ouverture d'une nouvelle fenêtre et ainsi de suite jusqu'à la rencontre des fichiers feuilles. Chaque fenêtre, au nom de la règle 4, est un agent d'édition. L'ouverture en cascade des fenêtres se traduit par la création d'agents d'édition liés par une relation père-fils.

Règle 9 : Si la spécification d'une commande (ou tâche élémentaire) implique des actions distribuées sur plusieurs agents, ceux-ci doivent être placés sous le contrôle d'un agent "ciment-syntaxique" qui synthétise les actions réparties en une unité d'information de plus haut niveau d'abstraction. L'exemple le plus courant de ce phénomène est la création d'exemplaires d'objet dans le domaine de l'édition graphique. Les classes d'objets (ou les concepts du domaine) sont regroupées dans une palette tandis que les exemplaires sont manipulés dans une fenêtre de travail. Typiquement, la création d'un exemplaire s'effectue en sélectionnant dans la palette la classe d'objet désirée puis en actionnant la souris dans la zone de travail pour spécifier les attributs de l'objet (localisation, taille, etc.). Dans cet exemple, l'action sur la palette détermine le nom de la commande (Créer-objet-de-classe-X) tandis que les actions dans la fenêtre de travail spécifient les paramètres de la commande. Ces actions réparties doivent être cimentées pour construire une action plus abstraite : une commande. Un agent PAC, père des agents qui interviennent dans la définition d'une commande, assure l'analyse syntaxique des actions sur ses agents fils. Ces derniers, après analyse lexicale des actions de l'utilisateur, transmettent des unités d'entrée à l'agent père qui effectue l'analyse syntaxique.

L'application systématique des règles énoncées jusqu'ici peuvent conduire à une prolifération d'agents. Ce phénomène est en accord avec le principe de la modularité, mais peut, selon les plates-formes de mise en œuvre, conduire à un excès de communication par message. Nous énonçons ci-dessous deux règles visant à éliminer des agents.

Règle 10 : Un agent et un agent fils unique peuvent être regroupés en un seul agent. Les fonctions des deux agents peuvent néanmoins faire l'objet de deux modules distincts. Cette règle ne doit être appliquée que si le père ne sera pas amené dans une version future du logiciel, à contrôler une seconde grappe d'agents.

Règle 11 : Un agent dont la fonction est réalisée par un objet de présentation ou d'interaction d'une boîte à outil peut être éliminé de l'arborescence. Il devient alors un composant de la Présentation de son agent père.

7.5.3. *Illustration : MATIS*

MATIS (*Multimodal Airline Travel Information System*) est un système d'information sur les transports aériens. La figure 7.9 montre l'essentiel de son interface graphique. Les figures 7.10 et 7.11 en illustrent l'architecture conceptuelle. Le Noyau Fonctionnel est une base de données sur les transports aériens accessible par des requêtes SQL. L'ANF joue ici le rôle de traducteur entre le formalisme SQL et la structure textuelle utilisée par le CD. De l'autre côté de l'arche, le CTP abstrait les informations spécifiées par l'utilisateur dans le formalisme du CD et vice versa. En l'occurrence, il est scindé en deux parties : l'une est dédiée à la présentation graphique et à l'interaction par manipulation directe sur les objets graphiques (partie englobée par Interface Builder), l'autre est dédiée à l'analyse des phrases en langage naturel (écrites ou parlées). Le Composant Bas Niveau d'Interaction désigne la plate-forme d'accueil logicielle et matérielle. Ce niveau regroupe les services d'acquisition, d'estampille et de répartition des événements mais aussi les objets d'interaction des boîtes à outils et les systèmes de reconnaissance spécialisés.

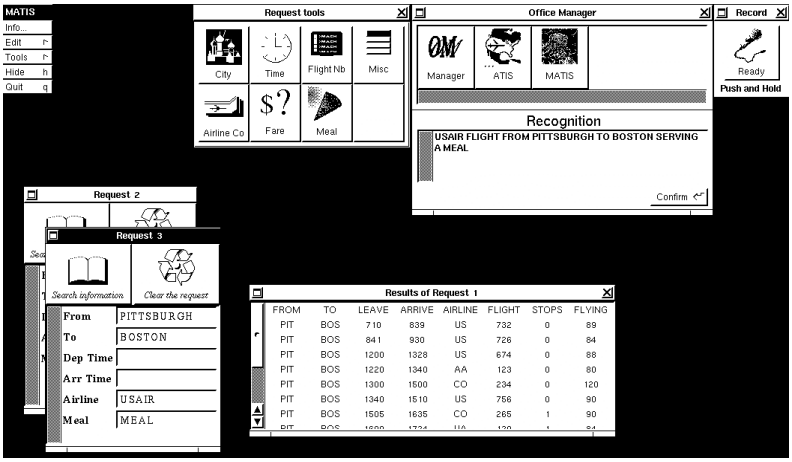

Figure 7.9. *Un exemple d'écran du système MATIS. En haut et au centre, la palette "Request Tools" permet de choisir via des menus des noms de ville ou de compagnie aérienne, une gamme de tarifs, etc. La fenêtre "Recognition" affiche la phrase que le système de reconnaissance de la parole vient de comprendre ; elle permet également à l'utilisateur de saisir au clavier une phrase en langage pseudo-naturel. L'icône en haut et à droite indique l'état du système de reconnaissance. En bas à gauche, deux formulaires de requête en cours d'édition. En bas et à droite, un tableau de résultats.*

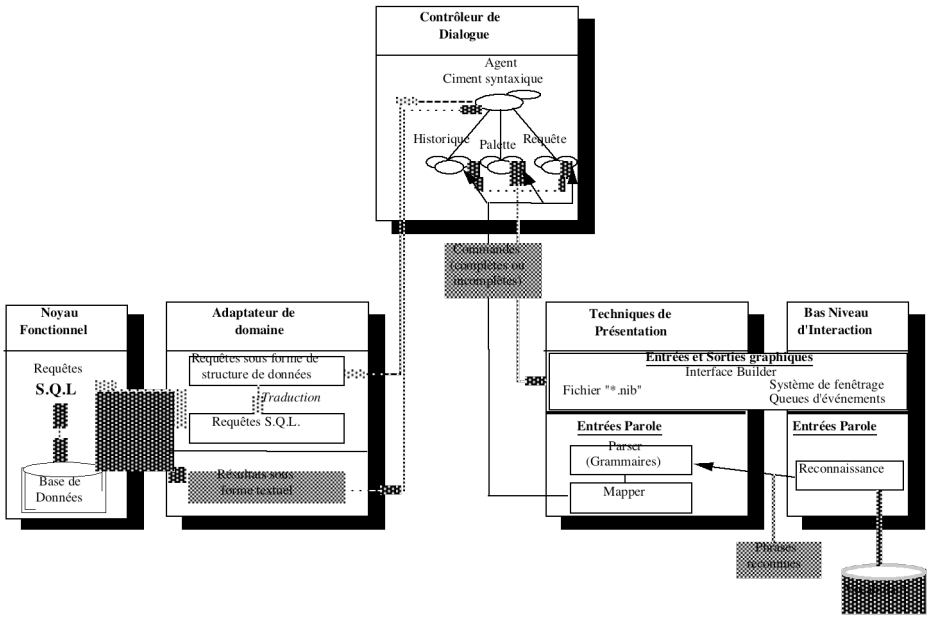

Figure 7.10. *Architecture PAC-Amodeus du système MATIS*

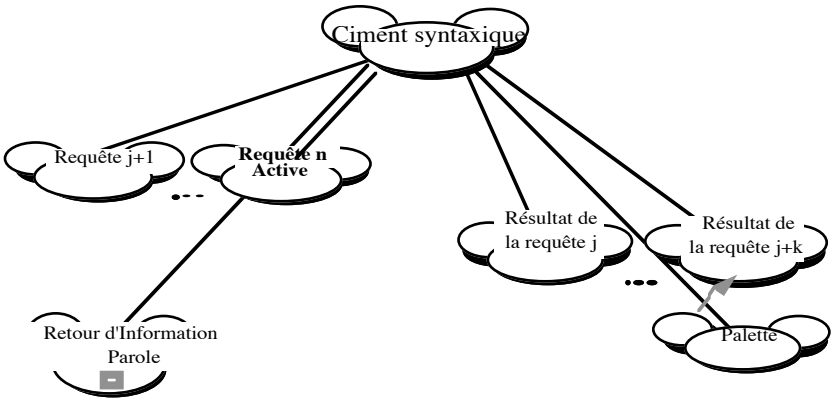

Figure 7.11. *Décomposition PAC du Contrôleur de Dialogue du système MATIS*

Application de la règle 1 : les agents *Requête* correspondent chacun à une requête en cours de spécification. Les agents *Résultat de requête* correspondent chacun à une réponse de requête (fenêtres "Results of Request"). L'agent *Retour d'Information* assure l'affichage d'un retour d'information sur tous les actes de parole affichée dans la fenêtre "Recognition" (analyse syntaxique de la phrase dictée) de même que l'édition d'une phrase saisie en langage pseudo-naturel.

Application de la règle 3 : L'agent *Palette* correspond à la palette d'outils comme les listes des compagnies aériennes et des villes (fenêtre "Requests Tools").

Application de la règle 9 : En racine de la hiérarchie, l'agent *Ciment syntaxique*, cimente les actions de l'utilisateur reçues et traitées par ses agents fils et communique avec l'ANF pour l'envoi d'une requête et la réception des réponses.

7.5.4. *PAC-Amodeus et interaction multimodale*

Les règles d'identification des agents et de simplification de la hiérarchie s'appliquent de la même manière au cas des interfaces multimodales. En particulier, un espace de travail, entité conceptuelle structurante, qu'il ait une présentation graphique ou vocale, doit être géré par un agent [NIG 93]. L'agent "vue multiple d'un même concept" est particulièrement pertinent dans le contexte de la multimodalité. Toutefois, l'infrastructure logicielle doit être enrichie d'un mécanisme de gestion des événements multimodaux. Dans PAC-Amodeus, ce mécanisme appelé "moteur de fusion" [NIG 95], ne fait pas partie de l'architecture conceptuelle mais de l'architecture implémentationnelle : le moteur est appelé par tout objet logiciel de la classe "agent".

7.5.5. *PAC-Amodeus et interaction multi-utilisateur : PAC**

PAC* [CAL 97] [COU 97] s'appuie sur PAC-Amodeus. Augmenté du trèfle des collecticiels [SAL 95a] [SAL 95b], pour l'aspect couverture fonctionnelle, il est aussi inspiré du modèle de Dewan [DEW 99] pour raisonner sur le partage et la réplication. La compétence locale d'un agent est affinée selon les trois dimensions du trèfle des collecticiels. Ainsi l'abstraction d'un agent se décompose en trois parties, dédiée chacune à l'une ou l'autre des facettes du trèfle : partie abstraite pour soutenir la production (notée A.Prod), partie abstraite pour la coordination (A.Coord) et partie abstraite pour la communication (A.Com). La même opération est appliquée aux facettes C et P. Typiquement, A.Prod a une présentation P.Prod reliée par un C.Prod. Comme le montre la figure 7.12, notre stratégie revient à tronçonner en trois tranches l'agent PAC originel.

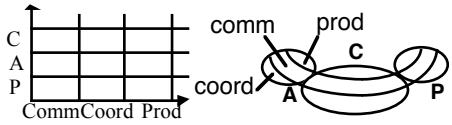

Figure 7.12. *Décomposition fonctionnelle d'un agent PAC dans PAC**

Ce "PAC Napolitain", de représentation compacte, masque, sous cette visualisation, les mécanismes sous-jacents de communication et de contrôle. Nous proposons trois vues sur ce modèle, pour rendre compte des stratégies envisageables.

– Version a) de la figure 7.13 : un agent poignée (un agent PAC restreint à sa fonction de contrôle) centralise les échanges entre facettes fonctionnelles. Toutes les relations de dépendance entre celles-ci y sont exprimées. Concentrées dans cet agent, elles facilitent évolutions et maintenance. Cet agent poignée, seule interface avec les agents pères, redirige les événements, suivant leur nature, vers l'une ou l'autre des facettes. Ces indirections simplifient certes l'implémentation, mais ne sont pas sans conséquence pour l'interface : traitements ralentis, temps de réponse augmentés, le prix classique de toute centralisation.

– b) : à l'opposé des versions centralisées, la version b) opte pour une répartition des dépendances entre facettes. Ces facettes communiquent entre elles sans intermédiaire : elles ciblent directement la facette pertinente. En découlent rapidité, efficacité, robustesse mais évolutivité plus limitée, de par l'éparpillement du code.

– c) : cette version hybride concilie les deux premières. Son intérêt réside dans la souplesse qu'elle procure : le spectre des combinaisons possibles est accessible dans sa globalité, au concepteur de jouer, par une stratégie fine, sur les avantages de chacune d'elles pour ne pas en cumuler les inconvénients. D'implémentation largement plus complexe, elle est, en pratique, plus délicate à mettre au point.

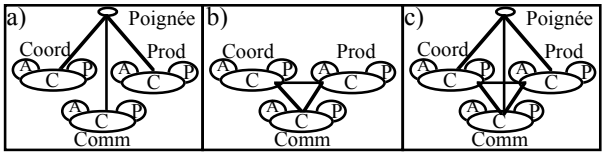

Figure 7.13. *La triple vue d'un agent PAC*.*

Chacune de ces versions présente ses avantages et inconvénients, qu'il faut peser dans le contexte d'une application donnée. C'est toujours par rapport à des critères que s'évaluent, de façon comparative, différentes alternatives. Typiquement, disposant d'un langage d'expression de contraintes, un agent poignée serait simple à réaliser. Nous avons jusqu'ici raisonné sans évoquer les problèmes de partage et de réplication propres aux systèmes répartis. Notre réponse s'inspire du modèle de Dewan. Le modèle de Dewan [DEW 99] peut se voir comme une généralisation du modèle à "fermeture éclair" de Patterson [PAT 94]. Comme le montre la Figure 7.14, un système est découpé en niveaux d'abstraction allant du niveau le plus haut de nature sémantique, au niveau le plus bas dépendant du matériel. Certaines couches sont partagées et constituent la base du système (niveaux S à N+1) jusqu'à un point de branchement (niveau N+1) à partir duquel les couches sont répliquées

sur chaque station utilisateur. La communication entre niveaux se fait par événements selon l'axe vertical entrée/sortie des niveaux d'abstraction mais aussi entre niveaux répliqués frères notamment pour la synchronisation d'état. Ce modèle offre un bon cadre de réflexion pour l'allocation des niveaux fonctionnels aux processus et permet de raisonner sur la granularité du parallélisme.

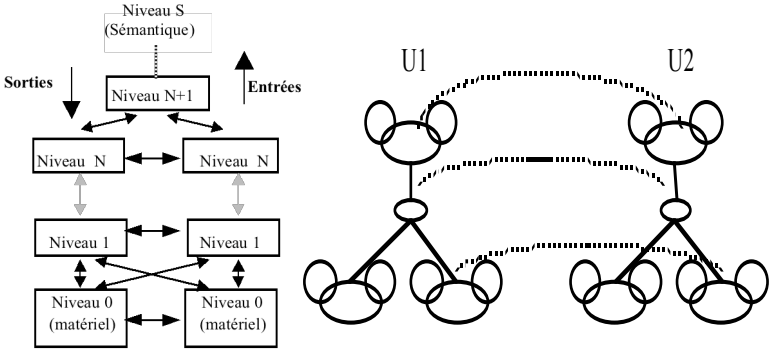

Figure 7.14. *A gauche, le modèle de Dewan. A droite, vue répartie sur deux stations U1 et U2 de la couche CD de notre instanciation du modèle de Dewan. Les autres couches de l'arche ne sont pas représentées.*

Dans PAC*, les niveaux sont ceux du modèle Arch (NF, ANF, CD, etc.). Nous préconisons de pousser le grain de partage, non pas au niveau de la couche comme chez Dewan, mais au niveau de l'agent ou d'une grappe d'agents. La figure 7.14 (partie droite) illustre une communication à différents niveaux d'abstraction au sein d'une couche donnée, ici celle du CD. Des liens horizontaux peuvent en effet, conformément au modèle de Dewan, être établis à différents niveaux dans la modélisation. Il faut néanmoins savoir que chaque communication signifie ouverture d'une connexion et donc, facturation associée. Par conséquent, même si la plate-forme d'accueil le permet, l'ouverture de connexions intempestives ne peut être décidée sans une étude de rentabilité : tout est question de compromis. Précisons que ces liens peuvent être permanents ou établis, au contraire, à la demande. Il faut, dans ce cas, spécifier ces conditions de connexion. Ces solutions techniques sont à discuter dans un contexte donné et ne peuvent, en aucun cas, être préconisées de façon systématique.

7.6. Conclusion

Nous avons présenté une revue de la terminologie liée à la conception logicielle d'un système d'information et plus généralement d'un système interactif. L'architecture logicielle est un ensemble de structures (conceptuelle, modulaire, de coordination et physique). Il n'y a donc pas une architecture pour un système donné

mais plusieurs correspondant à des perspectives différentes. Les travaux du domaine de l'IHM sur les architectures conceptuelles (structures conceptuelles) des systèmes interactifs, mono-utilisateur, multimodaux et collecticiels, ont ensuite été exposés. L'ubiquité nécessaire des systèmes d'information définit les verrous actuels pour la conception logicielle ; en particulier la mobilité des utilisateurs et la prise en compte du contexte physique d'interaction définissent des problèmes nouveaux qu'il convient d'étudier au niveau de l'architecture logicielle.

7.7. Annexe : Propriétés en Interaction Homme-Machine

Cette annexe consigne un ensemble de propriétés servant à caractériser la notion d'utilisabilité. Dans ce cadre, nous dirons que :

Utilisabilité = Souplesse + Robustesse

Où : la *Souplesse* (de l'interaction) exprime l'éventail des choix (pour l'utilisateur et le système) ; la *Robustesse* (de l'interaction) vise la prévention des erreurs, l'augmentation des chances de succès de l'utilisateur. Remarque : la facilité d'apprentissage n'est pas considérée.

Souplesse et robustesse sont déclinées en propriétés. Ces propriétés sont le résultat des travaux de l'équipe IIHM du laboratoire CLIPS en collaboration avec les membres du groupe de travail WG2.7 de l'IFIP et des partenaires du projet européen AMODEUS (http://www.mrc-cbu.cam.ac.uk/amodeus). Pour chacune d'elles, nous fournissons une définition complétée de remarques, d'exemples, de contre-exemples et, si possible, de métriques.

7.7.1. *Souplesse*

Atteignabilité

– Définition : capacité du système à permettre à l'utilisateur de naviguer dans l'ensemble des états observables du système. Un état q est atteignable à partir d'un état p s'il existe une suite de commandes { ci } qui font passer de l'état p à l'état q.

– Rem. : propriété non vérifiée si analyse de tâche ou analyse fonctionnelle défectueuse.

– Métrique : longueur de la trajectoire d'interaction entre p et q.

Non-préemption

– Déf. : le prochain but souhaité par l'utilisateur est directement atteignable. Pour l'utilisateur, il n'y a pas de contrainte dans la trajectoire interactionnelle.

– Contre-exemple : boîte de dialogue modale : à n'utiliser qu'à bon escient (cas des commandes irrévocables). Un système interruptible est, du point de vue utilisateur, non préemptif. Mais trop de liberté peut nuire : L'utilisateur risque d'être perdu. Penser au concept de tableau de bord qui indique à l'utilisateur sa localisation dans l'espace des tâches.

– Métrique : si la longueur de la trajectoire d'interaction entre l'état actuel et l'état souhaité vaut 1, alors la propriété de non préemption est vérifiée.

Préemption globale

– Déf. : interdiction, pour l'utilisateur, d'effectuer toute autre action que celle requise par le système (ici la station de travail).

Préemption locale

– Déf. : ne bloque qu'un fil de dialogue. Laisse l'utilisateur continuer les autres fils.

Préemption de ressources partagées par un utilisateur

– Ex. : collecticiels à "tour de parole" (*turn-taking*) : préemption du curseur partagé.

Interaction multifilaire

– Déf. : capacité du système à permettre la réalisation "simultanée" de plusieurs tâches.

– Rem. : analyse du caractère entrelacé ou parallèle à différents niveaux de granularité : niveau tâches au sein d'un logiciel donné, niveau actions (cf. les propriétés CARE).

– Ex. : édition de plusieurs documents à la fois.

– Métrique : nombre de tâches que l'on peut mener de manière entrelacée au regard de la charge cognitive.

Interaction multifilaire parallèle

– Déf. : parallélisme vrai entre plusieurs tâches. Peu souhaitable si les compétences de l'utilisateur n'est pas du niveau expert ("skill level", au sens de Rasmussen [RAS 86]).

Interaction multifilaire entrelacée

– Déf. : les tâches peuvent être simultanées au sens de l'utilisateur, mais à un instant donné, l'interaction est restreinte à une seule tâche.

Multiplicité du rendu (ou représentation multiple d'un même concept)

– Déf. : capacité du système à fournir plusieurs représentations pour un même concept.

– Rem. : (1) en sortie : formes différentes (par ex., pour le concept de température : un entier ou une représentation analogique comme un thermomètre. Contenus différents : le détail vs l'ensemble : attention aux discontinuités visuelles. Penser aux techniques "fisheye" ou holophrastiques. (2) En entrée : formes différentes : 6*4 et 24. (3) Principe d'égale opportunité : l'utilisateur choisit la nature de l'entrée, le système en déduit la sortie (principe des tableurs).

Réutilisabilité des données d'entrée et de sortie

– Déf. : les sorties du système peuvent être utilisées comme des données d'entrée (couper-coller). Les entrées de l'utilisateur peuvent être réutilisées par le système en sortie (valeurs par défaut).

– Rem. : attention aux effets de bord dus aux conversions de types de données. Cas du couper-coller entre deux éditeurs graphiques, l'un vectoriel et l'autre du niveau pixel.

Adaptabilité

– Déf. : personnalisation du système sur intervention explicite de l'utilisateur.

– Rem. : risque de perte de cohérence entre les systèmes d'une communauté d'utilisateurs sensés travailler ensemble. A considérer selon le degré de couplage des activités collectives. D'après [GRU 85], les utilisateurs ne modifient pas les valeurs par défaut qui viennent à la livraison du collecticiel. Leçon à retenir : bien étudier les valeurs par défaut initiales en fonction des catégories/rôles des futurs utilisateurs.

– Ex. : menus et formulaires d'options et de préférences. Macro d'encapsulation de commandes à caractère répétitif dont le niveau d'abstraction est trop bas. Toute donnée lexicale (ex. nom des commandes) doit être dans un fichier de ressources, donc pas dans le code source du logiciel. Mesure de vérification : produire le logiciel dans une autre langue sans le recompiler.

Adaptativité

– Déf. : capacité du système à s'adapter à l'utilisateur sans intervention explicite de l'utilisateur.

– Rem. : l'adaptativité s'appuie sur un modèle embarqué de l'utilisateur. Veiller à ce que le système ait un comportement prévisible. Ne pas surprendre l'utilisateur (caractère disruptif).

Plasticité

– Déf. : capacité du système à s'adapter aux variations des ressources interactionnelles, computationnelles, comunicationnelles et environnementales tout en conservant la continuité ergonomique [THE 99] [COU 98].

– Ex. : IHM d'un agenda sur PalmPilot et sur PC.

Migrabilité de tâche

– Déf. : capacité de délégation dynamique de tâches entre le système et l'utilisateur ou entre utilisateurs : changement dynamique de l'acteur(s) responsable(s) de l'accomplissement de la tâche.

– Rem. : manifestation à différents niveaux de granularité : (1) 6*4 et 24 (2) valeurs par défaut (3) détection de tâches répétitives puis prise en charge (Système Eager) (4) sauvegarde automatique de fichiers.

CARE (multimodalité)

– Déf. : complémentarité, Assignation, Redondance, Equivalence [COU 95] [NIG 94]. Caractérisation de la multimodalité offerte par un système. Modalité = <dispositif d'E/S, système représentationnel>. Complémentarité : plusieurs modalités distinctes sont nécessaires pour exprimer le but. Assignation : une seule modalité est disponible pour exprimer le but. Redondance : plusieurs modalités sont utilisables en "même temps" et expriment le même but. Equivalence : plusieurs modalités sont possibles pour exprimer le but. Une seule est utilisable à la fois. Notons que la redondance implique l'équivalence.

– Ex. : complémentarité : « mets ça là » vocal // geste. Redondance : « Montre-moi Grenoble » vocal // double-clic sur Grenoble affichée sur une carte.

CARE (collecticiel)

– Déf. : (1) appliqué aux rôles des acteurs d'un collecticiel [SAL 95a] : complémentarité de rôle (cas des jeux). Assignation de tâche à un rôle. (2) Appliqué aux moyens technologiques pour collaborer : CARE entre Fax, émail, Vphone etc.

7.7.2. *Robustesse*

Observabilité

– Déf. : capacité du système à rendre perceptible l'état pertinent du système. Capacité pour l'utilisateur à évaluer l'état actuel du système. L'utilisateur PEUT PERCEVOIR.

– Rem. : inspectabilité (browsability) : capacité pour l'utilisateur d'explorer l'état interne du système au moyen de commandes articulatoires (ou passives) (c'est-à-dire qui ne modifient pas l'état du NF) telles que zoom, défilement, etc.

– Contre-exemple : "defense in depth design" (Cf. [RAS 86]).

Observabilité publiée

– Déf. : capacité pour un utilisateur de rendre observables des variables d'état personnelles [SAL 95a] [SAL 95b].

– Rem. : exemples de variables d'état personnelles : présence, niveau de disponibilité. Une variable publiée peut être filtrée. Filtrage de variable : opération de transformation de la valeur de la variable visant à protéger l'espace privé. Un filtre ne doit pas être réversible.

Réciprocité

– Déf. : dans un collecticiel, capacité d'observation/inspection mutuelle des variables d'état personnelles.

– Ex. : « si je vous vois, vous me voyez. »

Réflexivité

– Déf. : capacité d'inspecter ou d'observer les variables d'état personnelles publiées à autrui.

– Ex. : vidéo miroir en vidéoconférence.

Insistance

– Déf. : capacité du système à forcer la perception de l'état du système. L'utilisateur DEVRA PERCEVOIR.

– Rem. : le retour d'information du système peut être pour un contexte donné : (1) éphémère (ex. audio, vidéo) ou non (ex. écrit statique), (2) évitable (retour visuel) ou inévitable (audio), (3) entretenu par le système (ex. clignotement) ou par l'utilisateur (maintien d'un bouton enfoncé qui minimise les oublis ; cf. [SEL 92]). Attributs perceptuels additifs ; l'intensité sonore et l'intensité des couleurs ont un effet additif. Le ton sonore et la teinte ne sont pas additifs. Rétro-action de groupe. Awareness : compréhension des activités d'autrui, fournissant à l'action individuelle un contexte situationnel collectif [DOU 92].

Honnêteté

– Déf. : capacité du système à rendre observable l'état du système sous une forme conforme à cet état ET qui engendre une interprétation correcte de la part de l'utilisateur. L'utilisateur AURA UN MODELE CORRECT de l'état du système.

– Rem. : WYSIWYG (What You See is What You Get), WYSIWIS (What You See Is What I See) et les versions relâchées. Distorsion des informations (ex. Les données linéaires ne doivent pas être présentées en deux dimensions [TUF 83]). Conformité de l'état interne et de la présentation pas toujours compatibles avec les temps de réponse attendus : utiliser un indicateur pour exprimer que l'information a changé et qu'elle n'est pas encore réactualisée dans le rendu. Intégrité des messages entre l'émetteur et le récepteur. Dans les formulaires, veiller à la formulation des unités de mesure et du format des données à saisir. Veiller à une terminologie précise en accord avec le métier, l'utilisateur, etc.

Honnêteté sociale

– Déf. : le système peut être honnête mais peut être détourné socialement.

– Ex. : enclencher son répondeur téléphonique pour simuler l'absence.

Curabilité

– Déf. : capacité pour l'utilisateur de corriger une situation non désirée.

– Rem. : (1) curabilité arrière : capacité de défaire. Le défaire de profondeur 1 est facile à réaliser : on ne modifie le NF qu'à l'interaction suivante. (2) Curabilité avant : reconnaissance de l'état actuel et capacité de négociation pour atteindre le but désiré = atteignabilité indispensable. Messages d'erreur explicatifs et correctifs. Principe de l'effort commensurable [DIX 93] : ce qui est difficile à défaire doit être difficile à faire (ex. destruction de fichier). Voir [JAM 96] pour une analyse détaillée.

Prévisibilité

– Déf. : capacité pour l'utilisateur de prévoir, pour un état donné, l'effet d'une action.

– Rem. : cohérence : conformité aux règles/usages [GRU 89]. Celles de l'utilisateur ne sont pas nécessairement celles du concepteur. Cohérence interne : cohérence lexicale, syntaxique, sémantique. Cohérence externe : conformité à des normes d'IHM, conformité à l'expérience dans le monde réel (analogie, métaphore). Retour d'information proactif : principe du "do-nothing" ou de résistance passive : éléments interdits en grisé. Prévisibilité et stabilité des temps de réponse : rassurer si temps de réponse long. Attention aux solutions système de type ramasse-miette à la volée sur la stabilité dans le temps de réponse. Régularité des médias continus. Attention aux limites de tolérance (ICS [BAR 85] [BAR 92] [BAR 94]).

Tolérance du rythme

– Déf. : l'utilisateur plutôt que le système décide quand il peut agir.

– Rem. : attention aux temporisations qui font sens (ex. durée de pression sur un dispositif, tels les téléphones portables et les distributeurs de boissons).

– Ex. : dans le cas de la saisie anticipée : nombre d'actions anticipables (noter que sous Word, ce nombre est variable.).

– Métrique : durée de tolérance.

Viscosité

– Déf. : l'action de l'utilisateur à un effet sur son plan de tâches ou, pour un collecticiel, sur celui des autres.

– Ex. : ajout d'une ligne dans un texte qui provoque des orphelins dans la pagination.

– Métrique : longueur de la trajectoire d'interaction nécessaire à la correction de l'effet de viscosité.

Rejouabilité

– Déf. : capacité pour l'utilisateur de rejouer des séquences informationnelles audio/vidéo.

– Rem. : s'appuie sur l'existence d'un historique. En communication médiatisée, la rejouabilité d'un message audio permet de réécouter [SAL 95a].

Révisabilité

– Déf. : capacité pour l'utilisateur de réviser un message avant de l'émettre.

– Rem. : la révisabilité implique la rejouabilité. Elle est une forme de curabilité.

7.8. Bibliographie

[AND 00] Anderson G., Graham N., Wright T., "Dragonfly: Linking Conceptual and Implementation Architectures of Multiuser Interactive Systems", *Proc. ICSE 2000*, ACM publ., p. 252-26, 2000.

[BAR 85] Barnard P.J., "Cognitive Resources and the Learning of Computer Dialogs", in *Interfacing Thought, Cognitive aspects of Human Computer Interaction*, J. M. Carroll (Ed.), MIT Press, p. 112-158, 1985.

[BAR 92] Barnard P.J. et May J., *Real time blending of data streams: a key problem for the cognitive modelling of user behaviour with multimodal systems*, Amodeus Project, Working Paper, UM/WP 26, 1992.

[BAR 94] Barnard P.J., May J. et Salber D., *Deixis and Points of View in Media Spaces*, Amodeus Project, Working Paper, UM/WP 19, 1994.

[BAS 98] Bass L., Clements P. et Kazman R., *Software Architecture in Practice*, Addison Wesley Publ., ISBN 0-201-19930-0, 1998.

[CAL 97] Calvary G., Coutaz J. et Nigay L., "From Single User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW", *Proceedings of CHI'97* (Atlanta, March 1997), ACM Press, p. 242-249.

[CAR 95] Carroll J.M., "Introduction: The Scenario Perspective on System Development", In *Scenario-Based Design: Envisioning Work and technology in System Development*, J.M. Carroll (Ed.), J.Wiley Publ., NY, 1995, p. 1-17.

[COU 90] Coutaz J., *Interface Homme-Ordinateur : Conception et Réalisation*, Dunod Publ., 1990.

[COU 95] Coutaz J. Nigay L., Salber D., Blandford A.E., May J. et Young R.M.Y., "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, p. 115-120, June 1995.

[COU 97] Coutaz J., "PAC-ing the Software Architecture of your User Interface", *In DSV-IS'97, 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems*, Springer Verlag Publ., p. 15-32, 1997.

[COU 98] Coutaz J., "Interfaces Homme-Machine : le Futur ne Manque pas d'Avenir", *Actes ERGO-IA'98*, Ed. ESTIA/ILS, p. 43-55, 1998.

[DER 94] Dery A.M. et Fornarino M., "Dépendances comportementales et modèle PAC", Actes des Journées IHM'94 , p. 87-94, 1994.

[DEW 99] Dewan P., "Architectures for Collaborative Applications", In *Computer Supported Co-operative Work*, M. Beaudouin-Lafon (Ed.), Wiley&Sons Pub., 1999.

[DIX 93] Dix A., Finlay J., Abowd G. et Beale R., *Human-Computer Interaction,* Prentice Hall, New York, 1993.

[DOU 92] Dourish P. et Bellotti V., "Awareness and Coordination in Shared Workspaces", In Proceedings Computer-Supported Cooperative Work, CSCW'92, ACM publ., p. 107-114.

[DUV 99] Duval T. et Nigay L., "Implémentation d'une application de simulation selon le modèle PAC-AMODEUS", Actes des Journées IHM'99, Cépaduès, p. 86-93, 1999.

[FAC 93] Faconti G., *Towards the Concept of Interactor*, SM/WP8, System Modelling, Working Paper 8, The Amodeus Project, Esprit Basic research Action 7040, 1993.

[FER 95] Ferber J., *Les systèmes multi-agents, Vers une intelligence collective*, InterEditions, Paris, 1995.

[GAM 95] Gamma E., Helm R., Johnson R. et Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, MA, 1995.

[GAR 93] Garlan D. et Shaw M., "An Introduction to Software Architecture", *In Advances in Software Engineering and Knowledge Engineering*, V. Ambriola and G. Tortora (Eds.), Vol. 1, World Scientific Publ., p. 1-39, 1993.

[GRA 96a] Graham T.C.N. et Urnes T., "Linguistic support for the evolutionary design of software architectures", *Proc. ICSE 18*, Berlin, Germany, p. 418-427, 25-29 Mars 1996.

[GRA 96b] Gram C. et Cockton G. (Eds.), *Design Principles for Interactive Software*. Chapman & Hall, 1996.

[GRU 89] Grudin J., "The Case Against User Interface Consistency", *Communication of the ACM*, 32(10), p. 1164-1173, 1989.

[GUI 95] Guittet L., *Contribution à l'Ingénierie des IHM - Théorie des Interacteurs et Architecture H4 dans le système NODAOO*, Thèse de l'Université de Poitiers, 1995.

[HIL 94] Hill R., Brinck T., Rohall S., Patterson J. et Wilner W., "The Rendez-vous language and architecture for constructing multi-user applications", *ACM Transactions on Computer-Human Interaction*, 1(2), p. 81-125, 1994.

[JAM 96] Jambon F., *Erreurs et Interruptions du Point de Vue de l'Ingénierie de l'Interaction Homme-Machine*, Thèse de l'Universitee Joseph Fourier, Grenoble 1, 1996.

[KAZ 94] Kazman R., Bass L., Abowd G. et Webb M., "SAAML A Method for Analyzing the Prperties of Software Architectures", *Proc. of ICSE-16*, p. 81-90, 1994.

[KRA 88] Krasner G. et Pope S., "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", *Journal of Object Oriented Programming*, p. 26-49, Août/Sept. 1988.

[MAC 90] Mackinlay J., Card S. et Robertson G., "A Semantic Analysis of the Design Space of input Devices", *Human-Computer Interaction*, 5(2&3), p. 145-190, 1990.

[MOS 94] Mostefai M., *Un modèle d'architecture orienté objet pour la conception de plates-formes de simulation interactive*, Thèse doctorat Univ. Lille I, 1994.

[NIG 93] Nigay L. et Coutaz J., "A design space for multimodal interfaces: concurrent processing and data fusion", *Proc. INTERCHI'93*, Amsterdam, p. 172-178, 24-29 Avril 1993.

[NIG 94] Nigay L., *Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales*, Thèse de l'Univ. Joseph Fourier, Grenoble 1, 1994.

[NIG 95] Nigay L. et Coutaz J., "A Generic Platform for Addressing the Multimodal Challenge", *Proceedings CHI'95*, Denver, p. 98-105, 7-11 Mai 1995.

[NIG 97] Nigay L. et Coutaz J., "Software architecture modelling: Bridging Two Worlds using Ergonomics and Software Properties", *In Formal Methods in Human-Computer Interaction*, P. Palanque & F. Paterno (Eds.), Springer-Verlag, London, p. 49-73, 1997.

[PAT 94] Patterson J.F., "A taxonomy of Architectures for Synchronous Groupware Applications", *Workshop on Software Architectures for Cooperative Systems, CSCW'94, ACM Conference on Computer Supported Cooperative Work*, Chappel Hill, NC, 1994.

[PFA 85] Pfaff G. (Ed.), *User Interface Management Systems*, Springer-Verlag, NY, 1985.

[PHI 99] Phillips G., *Architectures for Synchronous Groupware*, Department of Computing and Information Science, Queens's University, Ontario, Canada, March, 1999

[RAS 86] Rasmussen J., *Information processing and human-machine interaction, an approach to cognitive engineering*, Elsevier Science Publishing, 1986.

[SAL 95a] Salber D., *De l'interaction homme-machine individuelle aux systèmes multi-utilisateurs, L'exemple de la Communication Homme-Homme médiatisée*, Thèse de l'Université Joseph Fourier, Grenoble 1, 1995.

[SAL 95b] Salber D., Coutaz J., Decouchant D. et Riveill M., "De l'observabilité et de l'honnêteté : le cas du contrôle d'accès dans la Communication Homme-Homme Médiatisée", *Conférence sur l'Ingénierie des Interfaces Homme-Machine*, Toulouse, 1995.

[SEL 92] Sellen A., Kurtenbach G.P. et Buxton W., The "Prevention of Mode Errors through Sensory Feedback", *Human-Computer Interaction*, 7(2), p. 141-164, 1992.

[SHA 95] Shaw M. et Garlan D., *Software Architecture, Perspectives on an Emerging Discipline*, Prentice Hall, 1995.

[TAR 93] Tarby J.C., *Gestion Automatique du Dialogue Homme-Machine à partir de Spécifications Conceptuelles*, thèse de l'Université de Toulouse 1, 1993.

[TAR 97] Tarpin-Bernard F., *Travail Coopératif Asynchrone Assisté par Ordinateur : Approche AMF-C*, Thèse de doctorat, Ecole Centrale de Lyon, 1997.

[TAR 98] Tarpin-Bernard F., David B. et Primet P., "Framework and Patterns for synchronous groupware: AMF-C Approach", *Proc. IFIP 2.7 (13.4) Working Conference on Engineering for Human-Computer Interaction, EHCI'98*, Heraklion, Sept. 1998.

[TAY 96] Taylor R., Medvidovic N., Anderson K., Whitehead E., Robbins J., Nies K., Oreizy P., et Dubrow D., "A Component- and Message-based Architectural Style for GUI Software", *IEEE Trans. Software Engineering*, 22(6), p. 390-406, June 1996.

[THE 99] Thevenin D. et Coutaz J., *Plasticity of User Interfaces: Framework and Research Agenda*, *Proc. Interact'99*, Edinburgh, A. Sasse & C. Johnson (Eds.), IFIP IOS Press Publ.., p. 110-117, 1999.

[TUF 83] Tufte E., *The Visual Display of Quantitative Information*, Graphics Press, Cheshire Connecticut, 1983.

[UIMS 92] The UIMS Workshop Tool Developers : A Metamodel for the Runtime Architecture of an Interactive System, *SIGCHI Bulletin*, 24, 1, p. 32-37, Janvier 1992.