INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

|_|_|_|_|_|_|_|_|_|_|

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

École doctorale « Mathématiques, Informatique, Sciences et Technologies de l'Information »
Spécialité « Imagerie, Vision, Robotique »

présentée et soutenue publiquement par

Julien Letessier

le 23 octobre 2007

Vision pour l'interaction : une approche centrée utilisateur, orientée service

JURY composé de

Pr. Joëlle Coutaz	Présidente
Mr. Philip Gray	Rapporteur
Pr. Michel Beaudouin-Lafon	Rapporteur
Dr. Frédéric Vernier	Éxaminateur
Pr. James L. Crowley	Directeur de thèse
Dr. François Bérard	Co-encadrant

Thèse préparée dans les équipes PRIMA et IIHM du laboratoire LIG
LIG, équipe PRIMA — INRIA Rhône-Alpes — 655 av. de l'Europe — 38334 St. Ismier, France
LIG, équipe IIHM — 385, rue de la Bibliothèque — B.P. 53 — 38041 Grenoble, France

Résumé

À rédiger après les rapports.

Remerciements

À rédiger après les rapports.

Table des matières

Table des matières 7

Table des figures 11

1 Introduction : démocratiser la vision par ordinateur 15

2 Motivations et Objectifs 19

2.1 Intérêts de la vision pour l'interaction 20

2.1.1 Un moyen de perception généraliste 20

2.1.2 Un moyen d'interaction plus direct 22

2.1.3 Un support des interactions de nouvelle génération 26

2.1.4 Conclusion 28

2.2 Pourquoi la vision est elle sous-utilisée ? 28

2.2.1 Deux cas de non-utilisation de la vision 28

2.2.2 Défauts d'utilité 30

2.2.3 Défauts d'utilisabilité 31

2.2.4 Conclusions 32

2.3 Objectifs de la thèse 33

2.3.1 Approche conceptuelle 33

2.3.2 Mise en oeuvre, validation, et évaluation 33

3 Requis fonctionnels pour une boîte à outils utile et utilisable 35

3.1 Préambule : Vision dans les systèmes interactifs 36

3.1.1 Les systèmes interactifs post-WIMP 37

3.1.2 Faire le choix d'utiliser la vision 37

3.2 Des systèmes interactifs de nouvelle génération aux services perceptifs . . 39

3.2.1 Interfaces tangibles 40

3.2.2 Interfaces digitales 43

3.2.3 Systèmes collaboratifs 46

3.2.4 Nouvelles surfaces d'interaction 49

3.2.5 Interaction gestuelle 53

3.3 Synthèse : une taxonomie des besoins et services 55

3.3.1 Factorisation des services 55

3.3.2 Taxonomie des services de perception visuelle 58

3.3.3 Conclusion 62

4 Requis structurels pour une boîte à outils utile et utilisable 63
4.1 Définitions . . . 64
4.1.1 Service et architecture . . . 64
4.1.2 Critères d'évaluation des approches . . . 65
4.2 Familles d'approches existantes . . . 67
4.2.1 Bibliothèques classiques . . . 68
4.2.2 Approches monolithiques . . . 70
4.2.3 Approches à composants . . . 72
4.2.4 Approches orientées service . . . 74
4.3 Synthèse : Requis sur l'architecture . . . 79
4.3.1 Requis non fonctionnels . . . 79
4.3.2 Requis fonctionnels de l'architecture . . . 81
4.3.3 Choix de l'approche . . . 84
4.4 Conclusion . . . 85

5 Conception de la boîte à outils *gmlVision* 87
5.1 Notre approche . . . 88
5.1.1 Interface générique vs. interface *ad hoc* . . . 88
5.1.2 Recouvrement fonctionnel . . . 90
5.2 Architecture . . . 93
5.2.1 Choix de conception . . . 93
5.2.2 Interface externe orientée service . . . 94
5.2.3 Interface interne et support au développement des services . . . 97
5.2.4 Conclusions . . . 101
5.3 Un ensemble recouvrant de services . . . 102
5.3.1 Notations . . . 102
5.3.2 Services de suivi . . . 103
5.3.3 Services de capture . . . 110
5.3.4 Services support . . . 111
5.4 Conclusion . . . 115

6 Mise en œuvre de *gmlVision* 117
6.1 Architecture concrète de *gmlVision* . . . 118
6.1.1 *gmlBIP* : un middleware pour les services perceptifs . . . 118
6.1.2 Composants légers . . . 122
6.1.3 Inspection de services . . . 124
6.1.4 Conclusion . . . 125
6.2 Services fournis par *gmlVision* . . . 127
6.2.1 Le service *Finger Tracker* . . . 127
6.2.2 Widgets tactiles . . . 132
6.2.3 Suivi de visage . . . 136
6.2.4 Services de calibrage . . . 137
6.2.5 Services video . . . 140
6.2.6 Conclusions sur les services implémentés . . . 140
6.3 Conclusions . . . 141

7 Déploiement, validation et évaluation 143
7.1 Déploiement de prototypes 144
7.1.1 Historique du déploiement 145
7.1.2 Conclusions 157
7.2 Évaluation individuelle des services 159
7.2.1 gml.tracker.finger 160
7.2.2 gml.tracker.widgets 165
7.2.3 Comparaison avec d'autres surfaces tactiles 166
7.3 Évaluation de l'architecture 168
7.4 Conclusion 169

8 Conclusion : contributions, limitations et perspectives 171
8.1 Contributions 171
8.2 Limitations et perspectives 173
8.3 Conclusion 175

A Le protocole BIP : spécification et implémentation 177

B Outils avancés pour la vision par ordinateur 187
B.1 Métaprogrammation statique des primitives de traitement d'image 187
B.1.1 Techniques et outils existants 187
B.1.2 Utilisation de l'outil Lg dans *gmlVision* 187
B.2 vision par ordinateur sur cartes graphiques 188

C Formalisation des besoins rencontrés 189

D Glossaire des acronymes 193

E Dépendances logicielles de *gmlVision* 197

Bibliographie 199

Table des figures

2.1 Transition de l'interaction indirecte des interfaces graphiques classiques à différents paradigmes de manipulation directe. 24
2.2 Le *3D Blackboard*, un outil de dessin en trois dimensions. 25
2.3 Le projet *Illuminating Light*, une interface tangible pour la simulation d'un système optique. 25
2.4 Un exemple de transition de la manipulation directe dans le monde physique vers la manipulation directe dans le monde numérique : le montage video. 27
2.5 La caméra *EyeToy* et la *Wiimote*. 29
2.6 *DoodleDraw*, un système de dessin collaboratif sur surface augmentée fixe, portable, ou dirigeable. 30
2.7 *Interactive Public Ambient Displays* 30

3.1 Relations entre les quatre catégories d'utilisateurs et les différentes facettes de l'interface homme-machine de la boîte à outils *gmlVision*. 36
3.2 Le tableau interactif *DViT* conçu par Smart Technologies 38
3.3 Une table augmentée utilisant la réflexion totale frustrée. 39
3.4 Les cubes interactifs utilisés dans *Navigational Blocks* 41
3.5 *The Designer's Outpost* 42
3.6 La tablette tactile virtuelle *Visual Touchpad* 44
3.7 L'application *RoomPlanner* sur la table *DiamondTouch*. 45
3.8 Le jeu collaboratif multimodal *Caretta* 47
3.9 La *Table Magique* , un tableau blanc interactif pour l'augmentation des réunions informelles. 48
3.10 Le projet d'interaction sur surfaces dirigeables *Everywhere Displays* . . . 50
3.11 *i-LAND*, une salle de réunion augmentée offrant de nombreux dispositis d'interaction de nouvelle génération. 51
3.12 *Peephole Displays* 52
3.13 Le *PDS* (Portable Display Surface), une surface anodine augmentée. . . . 52
3.14 Deux projets utilisant les mouvements de la tête comme entrée 53
3.15 Deux projets utilisant la silhouette du corps comme entrée. 54
3.16 Liste des services rencontrés lors de l'état de l'art des systèmes interactifs de nouvelle génération. 55
3.17 Latence induite par les différents éléments d'un système interactif basé sur la vision par ordinateur. 56
3.18 Une taxonomie des services de perception visuelle pour l'interaction. . . . 59
3.19 Cartouches descriptifs d'un service ou d'un besoin. 61

4.1 Exemple de déploiement d'un système interactif utilisant PTK. 75

4.2 L'environnement intelligent *Easy Living*. 76
4.3 Paradigme abonnement / distribution d'événements. 82
4.4 Exemple de déploiement imposant l'isolation des services. 83

5.1 Le système *TAFFI*. 91
5.2 Cycle de développement adopté pour *gmlVision*. 92
5.3 Diagramme de classes UML de l'architecture interne à services de *gmlVision*. 102
5.4 Filtre infrarouge de la caméra video *Unibrain Fire-i 400*. 104

6.1 Schéma-bloc de la structure de la bibliothèque *gmlVision*. 117
6.2 Les modules du protocole `BIP/1.0`. 121
6.3 Code minimal permettant de se connecter à un service, utilisant le middleware *gmlBIP*. 122
6.4 Code minimal pour un service fonctionnel, utilisant le middleware *gmlBIP*. 123
6.5 L'inspecteur de *gmlVision* : paramètres du service. 125
6.6 L'inspecteur de *gmlVision* : canaux de communication. 126
6.7 L'inspecteur de *gmlVision* : images intermédiaires et résultats perceptuels. 126
6.8 L'inspecteur de *gmlVision* : graphiques. 126
6.9 L'architecture logicielle du service de suivi de doigts de *gmlVision*. . . . 128
6.10 Algorithme d'association de `tracker.finger`. 129
6.11 Génération des événements dans `tracker.finger` 130
6.12 L'interface de l'application *Finger Tracker.app*. 131
6.13 Le service perceptif *SPODS*, qui fournit des widgets digitaux. 132
6.14 Détection des occlusions pertinentes avec les *striplets*. 133
6.15 Boutons sensibles dans `tracker.widgets`. 134
6.16 Architecture du service `tracker.face`. 136
6.17 Formule de calibrage géométrique. 138
6.18 Mire de calibrage utilisée par `calibrator.automatic`, de dimension 4×4. 139

7.1 Historique du déploiement de prototypes utilisant *gmlVision*. 144
7.2 Le premier prototype *Sensitive Widgets*. 145
7.3 Les *widgets sensibles* utilisés dans le cadre de ContAct. 147
7.4 Le premier prototype *Finger Tracker* en fonctionnement. 148
7.5 Le prototype *FingerPDA*. 149
7.6 Architecture concrètre du prototype de *Table Magique* 152
7.7 Architecture du prototype *DoodleDraw* dans l'une des quatre conditions expérimentales. 153
7.8 L'application *DoodleDraw* en cours d'utilisation. 154
7.9 *TouchCalc*, un prototype qui utilise les striplets et SPODs. 155
7.10 Conditions du test d'utilisabilité de `gml.tracker.finger`. 160
7.11 Test d'utilisabilité de `gml.tracker.finger`. 161
7.12 Test d'utilisabilité de `gml.tracker.finger`. 162
7.13 Progression de la performance d'un utilisateur. 163
7.14 Mesure de latence pour `gml.tracker.finger` 164
7.15 La surface tactile *TouchLight* 166
7.16 Le clavier tactile de poche *Canesta*. 167
7.17 La surface interactive *SmartSkin* 167

7.18 La table *DiamondTouch* 167

A.1 Diagramme de classes UML de l'implémentation de référence de BIP/1.0, *gmlBIP*. 186

1 Introduction : démocratiser la vision par ordinateur

> *« The human-computer interface is easy to find in a gross way—just follow a data path outward from the computer's central processor until you stumble across a human being. (...) The key notion, perhaps, is that the user and the computer engage in a communicative dialogue whose purpose is the accomplishment of some sort of task. (...) All the mechanismes used in this dialogue constitute the interface. (...)*
> *At any point in the history of computer technology, there seem to be a prototypical user interface. A few years ago it was the teletypewriter ; currently it is the alphanumeric video-terminal. But the actual diversity is now much greater. »*
>
> *[Card et al., 1983]*

En préface de son ultime recueil de nouvelles, Philip K. Dick affirmait être « navré de devoir affirmer qu'en réalité, les auteurs de science-fiction ne savent rien du tout. Nous ne pouvons pas parler de science, car notre connaissance en est limitée et officieuse ; et en général, notre fiction est épouvantable. » [Dick, 1985]. En adaptant sa nouvelle *Minority Report* au cinéma, l'équipe de Steven Spielberg a pourtant parfois choisi de substituer le réalisme à l'imaginaire. Dans l'une des scènes-clé du film, le personnage de Jon Anderton interagit avec l'écran géant et transparent d'un ordinateur : il déplace, découpe, et navigue dans des séquences d'images en effectuant des gestes fluides avec ses mains. Ce type d'interface, loin de relever de la fiction, est visiblement l'héritier des travaux fondateurs de Myron Krueger sur la vision artificielle pour l'interaction Homme-machine [Krueger et al., 1985], d'ailleurs contemporains de Dick.

Chapitre 2

Notre intuition est que si les interfaces utilisateurs fondées sur des techniques de vision par ordinateur ne sont pas forcément appelées a devenir « prototypiques » au sens de Card, elles devraient un jour prendre une place significative dans l'ensemble varié des technologies utilisées pour l'interaction. Cette intuition est étayée, d'un point de vue théorique, par la possibilité de concevoir une interaction plus directe dans le sens où les intermédiaires habituels entre l'humain et la machine sont supprimés (clavier, souris, etc.). La vision peut également être considérée comme un moyen de perception généraliste, car elle permet de réaliser des types d'interaction variés avec un unique dispositif physique. D'un point de vue pragmatique, nous constatons que depuis plusieurs décennies de nombreux systèmes qui s'appuie sur la vision par ordinateur ont été construits par des chercheurs.

Pourtant, hors de la communauté scientifique, les interfaces utilisateur utilisent peu la vision. À notre connaissance lee seul produit répandu est la caméra *EyeToy* de

Sony et la *Wiimote* de Nintendo qui penvent être connectée à une console de jeux pour fournir une forme d'interaction gestuelle très rudimentaire.

La vision par ordinateur est donc pertinente comme support de l'interaction Homme-machine, mais son utilisation est encore marginale. Nous postulons que cette situation est due au manque d'outils logiciels de vision par ordinateur adaptés aux besoins des concepteurs de systèmes interactifs. En effet, les outils conçus par des spécialistes en vision sont généralement trop complexes à utiliser et ne fournissent pas des services pertinents pour l'interaction ; les outils conçus par les spécialistes en interaction sont souvent trop limités d'un point de vue fonctionnel.

Notre objectif est de contribuer à la démocratisation de la vision pour l'interaction en identifiant des principes, techniques et outils permettant de bâtir une boîte à outils utile et utilisable.

Chapitre 3

Nous faisons le constat pragmatique que cette démocratisation passe par la coopération entre trois types d'acteurs qui ont des profils bien différents. Pour simplifier le discours, nous donnons un prénom à chaque type d'acteur : Laurence est le concepteur et développeur de nouvelles interactions. En général elle n'a pas d'expertise dans le domaine de la vision par ordinateur. Stanislas est l'expert en vision par ordinateur. En général il n'est pas familier des problèmes liés aux facteurs humain. Enfin, Patrick est le développeur de services logiciels. Son expertise est centrée sur le génie logiciel mais il est familié des concepts et du vocabulaire spécifiques à la vision par ordinateur et à l'interaction Homme-machine. Notre travail vise en particulier à faciliter la communication et la coopération entre ces trois types d'acteurs.

À cet effet, nous proposons une façon de définir le problème sous la forme de contrats de perception. Ces contrats peuvent être exprimés comme un besoin de Laurence ou comme un service offert par Patrick. Nous proposons une notation compacte pour exprimer ces contrats. Cette notation vise à être assez spécifique pour permettre l'expression de contraintes pertinentes en vision par ordinateur, mais assez générale pour être aisément exploitable par Laurence qui n'est pas experte en vision par ordinateur.

Nous utilisons cette notation pour définir un ensemble de services de perception visuelle qui couvre la totalité des besoins des systèmes interactifs étudiés dans notre état de l'art. Cette état de l'art concerne les systèmes interactifs innovants qui mettent en oeuvre, ou pourrait mettre en oeuvre, la vision par ordinateur.

Chapitre 4

Une fois la communication possible entre les différents acteurs, il est nécessaire de définir une interface (de programmation) pour les services de perception et une structure pour la boîte à outils qui contient ces services. Nous réalisons un état de l'art des bibliothèques de services de vision existantes et des catégories d'architectures logicielles correspondantes, que nous comparons à un ensemble de critères centrés sur l'utilisateur.

Cet état de l'art nous amène à définir quatre propriétés qu'une boîte à outils de vision pour l'interaction doit satisfaire pour être utile et utilisable : asynchronisme (le paradigme de communication par évènements doit être employé), abstraction (l'emploi de la vision doit être invisible pour l'utilisateur), isolation (un service doit être fonctionnellement indépendant du reste du système), et contrat (un service doit s'engager à remplir un contrat de perception donné).

Chapitre 5

Nous proposons pour satisfaire ces requis structurels d'adopter une architecture orientée service. Chaque service est une entité isolée du reste du système : une « boîte noire » pourvue de canaux de communications et qui véhicule de manière asynchrone des événements discrets dans une représentation simple.

Le cadre conceptuel fourni par l'architecture à services nous permet de définir l'interface des services qui composeront notre boîte à outils. Pour déterminer quels services spécifiques fournir, nous adoptons une démarche empirique et itérative inspirée de

modèles classiques en génie logiciel. Cette démarche est centrée sur les besoins tout en s'adaptant aux limitations inhérentes à la vision par ordinateur. En appliquant cette démarche nous déterminons un ensemble de services qui recouvre l'ensemble des besoins identifiés : services pour les interfaces tactiles, les interfaces tangibles, etc.

Chapitre 6

Pour implémenter le support structurel de notre boîte à outils *gmlVision*, nous choisissons de définir le protocole de communication `BIP/1.0`. Ce protocole extensible est centré sur la simplicité d'interopération. Nous l'implémentons dans l'intergiciel *gmlBIP*.

Nous utilisons *gmlBIP* en tant qu'interfaces entre notre boîte à outils et ses utilisateurs. Nous réalisons certains services que nous avons conçus, en particulier : un suivi de doigts et un détecteur d'occlusion qui permettent de réaliser des interfaces tactiles. Nous implémentons également des services de support (acquisition video, calibrage géométrique automatisé) qui contribuent à l'utilisabilité de *gmlVision*.

Chapitre 7

Notre boîte à outils a été utilisée tout au long de sa conception et de son développement pour réaliser une variété de prototypes de systèmes interactifs. Nous exploitons rétrospectivement ces différentes expériences pour en tirer des enseignements sur *gmlVision*. Après une évaluation quantitative et qualitative, les services individuels apparaissent utilisables et performants. Par contre, il semble nécessaire d'identifier des pratiques et des patrons de conceptions de systèmes interactifs utilisant la vision pour fournir le support correspondant dans *gmlVision*.

Chapitre 8

Nous concluons ce rapport en résumant nos contributions, et identifiant certaines limitations et en proposant un ensemble de perspectives ouvertes par nos travaux.

2 Motivations et Objectifs

> *« Computer vision is a very difficult problem still far from being solved in the general case after several decades of determined research, largely driven by a few main applications. [...] Compared to speech recognition technology, which has seen years of commercial viability and has been improving steadily for decades, computer vision technology for HCI is still in the Stone Age. »*
>
> *[Turk, 2004]*

La vision par ordinateur est une source potentielle de nombreuses avancées en interaction Homme-machine, c'est ce que nous visons à montrer dans la première partie de ce chapitre. Pour cela nous nous interessons aux systèmes interactifs qui exploitent la vision, ou pourraient en bénéficier. Dans certaines situations la vision semble être la seule approche capable de fournir une solution au problème d'interaction. Dans d'autres situations, la vision apporte un avantage par rapport aux approches traditionnelles. Les systèmes décrit dans ce chapitre motivent l'intérêt de la vision par ordinateur dans la communauté recherche en interactions homme-machine ainsi que dans le cadre d'applications commerciales.

Malgré le potentiel de la vision par ordinateur, les chercheurs qui conçoivent et réalisent les systèmes interactifs font rarement le choix de la vision. Dans la rares situations où la vision est utilisée, sa mise en oeuvre est décrite en détail : il est fréquent de rencontrer des projets où la plus grande partie des efforts et du volume des publications est consacrée à la conception d'algorithmes de vision au détriment de l'évaluation du système du point de vue de l'utilisateur.

Il apparaît que l'existant en terme de bibliothèques logicielles de services de vision par ordinateur pour l'interaction Homme-machine n'est pas satisfaisant. Nous développons en fin de chapitre notre hypothèse selon laquelle les bibliothèques de vision ne sont pas **utiles** et **utilisables** du point de vue des concepteurs et développeurs de systèmes interactifs. Cette hypothèse est justifiée suivant les deux axes. D'une part, la vision ne fournit pas encore de solution à tous les problèmes de perception, et se heurte à la concurrence de dispositifs d'entrée spécifiquement adaptés et prêts à l'emploi, ce qui limite son utilité. D'autre part, soit les services développés ne sont pas mis en commun, soit ils sont mal ciblées, ce qui limite l'utilisabilité de la vision.

Pour conclure ce chapitre, nous présentons les objectifs de ce travail de thèse en section 2.3 : déterminer les requis d'utilité et d'utilisabité d'une « bonne » boîte à outils de vision pour l'interaction, proposer une approche technique adaptée, l'implémenter, et l'évaluer.

2.1 Intérêts de la vision pour l'interaction

Les machines disposent de facultés qui leur permettent de percevoir leur environnement et l'activité humaine, de l'interpréter, et de réagir en fonction de ces informations. Ces facultés sont fournies par des instruments matériels et logiciels de perception, de raisonnement, et d'action ; certains sont spécifiques aux machines, d'autres tendent à reproduire un équivalent chez l'homme.

Nous nous interessons à la perception, qui permet l'interaction « en entrée » entre un humain et un ordinateur. Elle est typiquement implémentée à l'aide de *dispositifs d'entrée* comme un clavier ou une souris ; qui détectent et mesurent certaines actions bien définies de l'utilisateur ; elle peut aussi l'être par un logiciel qui observe l'utilisateur par le truchement d'une caméra video : c'est ce que nous nommons *vision pour l'interaction*.

Afin d'illustrer l'intérêt de ce moyen de perception, nous le comparons aux dispositifs d'entrée traditionnels : contrairement à ces derniers, il peut être qualifié de *généraliste*, car le même matériel permet à l'utilisateur d'interagir de multiples façons — c'est-à-dire en utilisant de multiples techniques d'interaction. Nous montrons ensuite qu'il permet de rapprocher l'utilisateur des objets numériques qu'il manipule en rendant l'interaction plus directe, en particulier dans le cas des surfaces interactives. Pour finir, nous montrons qu'il est adapté aux interactions de « nouvelle génération », celles qui s'affranchissent des contraintes de l'informatique traditionnelle.

Notre objectif ici est simplement de donner un point de vue trés général de l'interêt de la vision pour l'interaction. En particulier nous ne cherchons pas à démontrer point par point son intérêt vis à vis d'autres technologies pour l'implémentation d'applications précises. Nous faisons l'hypothèse que la vision a un intérêt en interaction Homme-machine, hypothèse supportée par les nombreux exemples de systèmes interactifs ayant recours à la vision par ordinateur détaillés au chapitre suivant. Partant de cette hypothèse, notre objet est de faciliter sa mise en oeuvre dans les systèmes interactifs.

2.1.1 Un moyen de perception généraliste

Les pionniers ont conçu des outils de perception dédiés à une tâche particulière : le clavier pour la saisie de texte, et la souris pour la désignation d'un centre d'intérêt sur un écran [Engelbart et al., 1967]. C'est grâce aux informations fournies par ces organes que l'action d'un utilisateur est traditionnellement perçue. Mais la variété des tâches d'interaction croissant, le clavier et la souris ne sont plus toujours adaptés à l'interaction.

2.1.1.1 Défauts des dispositifs d'entrée traditionnels

Dans le cadre de l'informatique généraliste, une même machine est utilisée pour réaliser des tâches diverses, allant des tâches pour lesquelles les outils ont été initialement conçus (la bureautique, c'est-à-dire la manipulation de documents essentiellement textuels) à des tâches complexes sur le plan de l'interaction (comme le montage video, ou la modélisation de scènes tridimensionnelles). Cependant, le nombre d'outils de perception disponibles, ainsi que leur capacité d'expression, est limité : il est difficile d'imaginer un ordinateur équipé d'un périphérique d'entrée pour chacune des tâches possibles. Un compromis est donc nécessaire entre, d'une part, l'adaptation des outils choisis aux tâches d'intérêt, et d'autre part, la généralité de ces outils.

Le choix traditionnel est celui de l'ordinateur personnel (ou « Personal Computer », PC), aujourd'hui pervasif. En simplifiant, il est composé d'une unité de calcul généraliste pour le traitement, d'un affichage graphique comme moyen d'action, et d'un clavier et d'une souris comme moyen de perception. Dans le cas où cet ensemble est trop limitatif, il lui est adjoint un organe supplémentaire, mieux adapté. Pour la perception, ce seront des dispositifs d'entrée (*input devices*) spécifiques à un ensemble

de tâches. Une tablette graphique est utilisée pour le dessin, une spacemouse pour la modélisation d'objets en trois dimensions, un joystick pour la simulation, un table de mixage ou de montage pour la création de média.

Spécialisation des techniques d'interaction.

Dans le cas ou seuls les dispositifs généralistes (clavier, souris) sont utilisés pour une tâche d'interaction nouvelle, la complexité est reportée sur l'utilisateur. Prenons un exemple d'interaction dans un espace en trois dimensions. Une des tâches centrales est le *docking* : il s'agit de placer un objet relativement à un autre suivant les six degrés de liberté. De nombreuses techniques de sélection et de manipulation dans l'espace à la souris ont été conçues et comparées pour accomplir cette tâche. Par exemple dans [Nielson et Olsen, 1987] les mouvements verticaux et horizontaux de la souris sont interprétés comme des mouvements selon x et y, et les mouvements en diagonale comme des mouvements selon z. Cette interaction fait partie d'une famille appelée « contrôleurs virtuels » par Nielson : ils consistent à permettre à l'utilisateur de contrôler plus de degrés de libertés que le dispositif n'en fournit naturellement. En suivante cette approche, déplacer un objet dans l'espace en translation et en rotation — la tâche de *docking* — correspond alors à six acquisitions de cible complexes, alors qu'une seule acquisition suffit avec un objet du monde physique.

Spécialisation du dispositif d'entrée.

Il est possible de réduire cette complexité en proposant un dispositif spécialisé à l'utilisateur. En restant dans le contexte de l'interaction dans un espace en trois dimensions, le *GlobeFish* est un périphérique d'entrée à deux fois trois degrés de liberté (trois en rotation isotonique libre, trois en translation isométrique) qui est conçu pour réaliser le même type de tâche [Froehlich et al., 2006]. Ces dispositifs sont conçus et évalués essentiellement pour optimiser leurs performances en termes d'accomplissement de la tâche (par exemple des tâches de docking). Mais ils ne tiennent pas compte d'autre critères de bonne conception d'un périphérique d'entrée tels que ceux présentés dans [Balakrishnan et al., 1997]. En particulier, pour le GlobeFish, la relation entre les degrés de liberté du périphérique et ceux de l'objet virtuel manipulé est complexe. D'après Balakrishnan, la spécialisation de ce type de dispositif leur fait naturellement perdre en généralité : ces dispositifs sont mal adaptés aux tâches classiques de pointage en 2 dimensions.

Conséquences.

Les concepteurs d'applications, et *in fine* les utilisateurs, parviennent donc au compromis décrit ci-dessus, que nous pouvons reformuler de la façon suivante. En conservant les outils habituels (clavier et souris), au pouvoir expressif limité, il est nécessaire de concevoir et d'apprendre une technique d'interaction potentiellement complexe et peu naturelle. L'alternative est de concevoir et d'apprendre à utiliser un dispositif physique adapté précisément à la tâche d'intérêt, mais qui ne pourra facilement être général.

La spécialisation des techniques d'interaction, comme celle des dispositifs d'entrée, a donc un coût pour les développeurs comme pour les utilisateurs finaux.

2.1.1.2 Perception généraliste en vision

La vision par ordinateur dématérialise les dispositifs de perception en séparant le capteur et le processus de perception ; c'est-à-dire en déportant la perception dans la partie cognitive du système. La vision par ordinateur permet d'implémenter plusieurs besoins de perception spécialisée grâce à un dispositif physique unique (la caméra). C'est pourquoi nous qualifions la vision de mode de perception *généraliste*, même si chaque interaction produite par un processus de vision peut être spécialisée.

Par exemple, la surface électronique *DiamondTouch* [Dietz et Leigh, 2001] est le support de l'application *RoomPlanner* [Wu et Balakrishnan, 2003] : elle permet le suivi

d'un doigt par utilisateur. La surface électronique *e-Board* [Sugimoto et al., 2001] est utilisée pour implémenter l'application collaborative *Caretta* [Sugimoto et al., 2004] : elle permet la localisation de *phycons* (icônes physiques) équipées d'une puce RFID. La vision par ordinateur permet d'implémenter les deux besoins de perceptions de ces systèmes. En pratique, elle permet donc de remplacer les deux surfaces électroniques employées par un seul dispositif. Outre le coût inférieur de cette approche, elle rend possible une reconfiguration aisée du mode d'interaction en fonction du besoin.

Cet caractéristique généraliste de la vision est liée à la richesse de l'information contenue dans un flux vidéo. À la différence des dispositifs classiques, conçus de façon ad-hoc pour percevoir uniquement une information ciblée, la vision par ordinateur procède par filtrage d'une information très riche de façon à isoler l'information ciblée. Cette caractéristique fondamentale est à la fois une source de difficulté (le filtrage est un processus complexe à concevoir et à exécuter) mais aussi l'intérêt principal de la vision par ordinateur car elle lui confère un grand potentiel en terme de possibilité de création de nouvelles formes d'interaction.

2.1.2 Un moyen d'interaction plus direct

L'utilisation de la vision permet de ne rien interposer entre l'utilisateur et l'objet de l'interaction : au contraire des dispositifs d'entrée « physiques », elle permet donc à l'utilisateur d'exploiter dans le monde numérique son expérience de l'interaction dans le monde réel.

Dans cette section, après avoir présenté la notion d'interaction indirecte, nous montrons que la vision permet de réaliser des interactions plus directes qu'avec des dispositifs d'entrée classiques. Nous montrons ensuite que c'est la seule technique qui permet à de réaliser des interactions vraiment directes, et de *venir tel quel* pour interagir. Nous illustrons pour finir ces avantages intrinsèques de la vision dans le cas des surfaces interactives.

2.1.2.1 Interaction indirecte

Expliquons tout d'abord en quoi l'interaction classique (à la souris) est indirecte.

Prenons comme exemple le montage vidéo assistés par ordinateur. Il s'agit de la transposition dans le monde numérique d'une activité réalisée auparavant mécaniquement : historiquement, les bandes video analogiques étaient assemblées à la main, puis à l'aide de machines semi-automatiques. La réalisation de cette tâches avec l'assistance d'un ordinateur apporte une valeur ajoutée. D'une part, bien entendu, il s'agit des avantages liés à l'utilisation d'un support numérique : qualité du produit fini, possibilité de réalisation d'effets spéciaux, et précision du montage. D'autre part, le passage à l'édition virtuelle, dont les objectifs et les techniques sont décrits dans [Mackay et Davenport, 1989], permet de circonvenir pour la réalisation de la tâche une limitation importation de l'édition analogique : le temps du montage, auparavant très long, est réduit à un ordre de grandeur comparable à celui du processus créatif de décision du montage.

Quel que soit le dispositif d'entrée conventionnel utilisé, l'utilisateur manipule cependant l'outil de perception (le clavier poste de montage, puis la souris) et non l'information (les chutes video) elle-même. Malgré l'introduction du paradigme dit de manipulation directe [Shneiderman, 1983], l'interaction est indirecte. L'interaction avec les objets virtuels *via* des périphériques ne peut offrir la même affordance qu'avec des objets réels. Les dispositifs d'entrée conventionnels peuvent donc constituer un frein à l'apprentissage, la créativité et la performance de l'utilisateur.

2.1.2.2 Interaction directe

Nous nous intéressons ici en particulier aux *surfaces interactives*, c'est-à-dire les systèmes où l'utilisateur interagit sur le lieu de l'affichage, par exemple un *tablet PC*.

Dans ces systèmes, les lieu d'action et de perception de la machine et ceux de l'utilisateur sont co-localisés et confondus.

Souvent, dans ce type de système, l'interaction à la souris cède la place à une manipulation plus naturelle : stylet, doigts, ou objets divers permettent d'agir sur l'information numérique.

Dans ce contexte, des techniques de manipulation « plus » directe permettent de se rapprocher de l'affordance des techniques d'interaction dans le monde réel (figure 2.1 page suivante). La manipulation classique à la souris, dite directe [Shneiderman, 1983], est en fait une interaction indirecte : le dispositif d'entrée contitue un intermédiaire. En la remplaçant par les doigts de l'utilisateur (interface digitale), ou en matérialisant un contrôle logique (interface saisissable), l'interaction devient plus directe. Ce type d'interface peut être réalisé en vision ; nous en donnons des exemples dans les paragraphes suivants.

Enfin, lorsque le membre de l'utilisateur, le dispositif physique, et le dispositif logique qu'il contrôle sont fusionnés, on obtient une interaction sans aucune indirection. C'est le cas de la *Perceptual Window* (figure 3.14 page 53) et du *3D Blackboard* (figure 2.2 page 25). À notre connaissance, la vision est la seule technologie qui permette de réaliser ce type d'interface.

2.1.2.3 « Venez tels quels »

Cette expression est attribuée à Myron Krueger (de l'anglais *come as you are*). Elle traduit une seconde propriété intrinsèque de l'emploi de la vision : pour créer l'interaction, il n'est nécesssaire d'équiper ni l'utilisateur, ni le lieu de l'interaction.

Le cadre de recherche des interactions de nouvelle génération impose un relâchement des contraintes de l'informatique conventionnelle. L'environnement peut être modifié, l'action n'a plus nécessairement lieu devant un écran, dans un bureau : il peut s'agir d'un mur, d'une salle de réunion, d'un lieu de passage. Plusieurs utilisateur peuvent interagir avec le système interactif, et leur nombre peut varier dans le temps.

Pour ces raisons, il est souvent difficile, et parfois impossible de les équiper pour la tâche d'interaction qu'il vont réaliser (par exemple en leur confiant un dispositif particulier). Les utilisateurs doivent donc pouvoir venir tels quels : la vision est donc particulièrement adaptée à cette nouvelle contrainte, puisque l'outil de perception (la caméra) est indépendant de l'utilisateur.

Dans le cas (récurrent) où il n'est pas non plus acceptable d'instrumenter l'environnement, la vision est, là encore, la seule technologie de perception qui peut implémenter le *venir tel quel*.

2.1.2.4 Cas des surfaces interactives en vision

Les surfaces interactives permettent le retour à l'interaction naturelle, telle qu'elle a lieu avec les objets du monde réel. Celles utilisant la vision par ordinateur sont souvent appelées « surfaces augmentées » ou « bureaux augmentés » : le concepteur améliore leurs fonctionnalités en les rendant interactives, grâce à la projection video et un système perceptif. Selon si la tâche d'interaction concerne des objets du monde réel ou virtuel, le système obtenu sera appelé un système de réalité augmentée, respectivement de virtualité augmentée. Introduisons brièvement deux exemples de systèmes interactif illustrant l'intérêt de la vision pour ces deux types de systèmes interactifs.

Illuminating Light [Underkoffler et Ishii, 1998] est un système interactif dont l'interface est saisissable (figure 2.3 page 25), sa réalisation s'appuie sur un système de vision. L'évaluation de ce système indiquent que, pour la tâche étudiée, le système est satisfaisant pour les utilisateurs. Ils sont même plus performants que sur le système réel et ce même dans le cas d'utilisateurs experts. Remarquons par ailleurs que

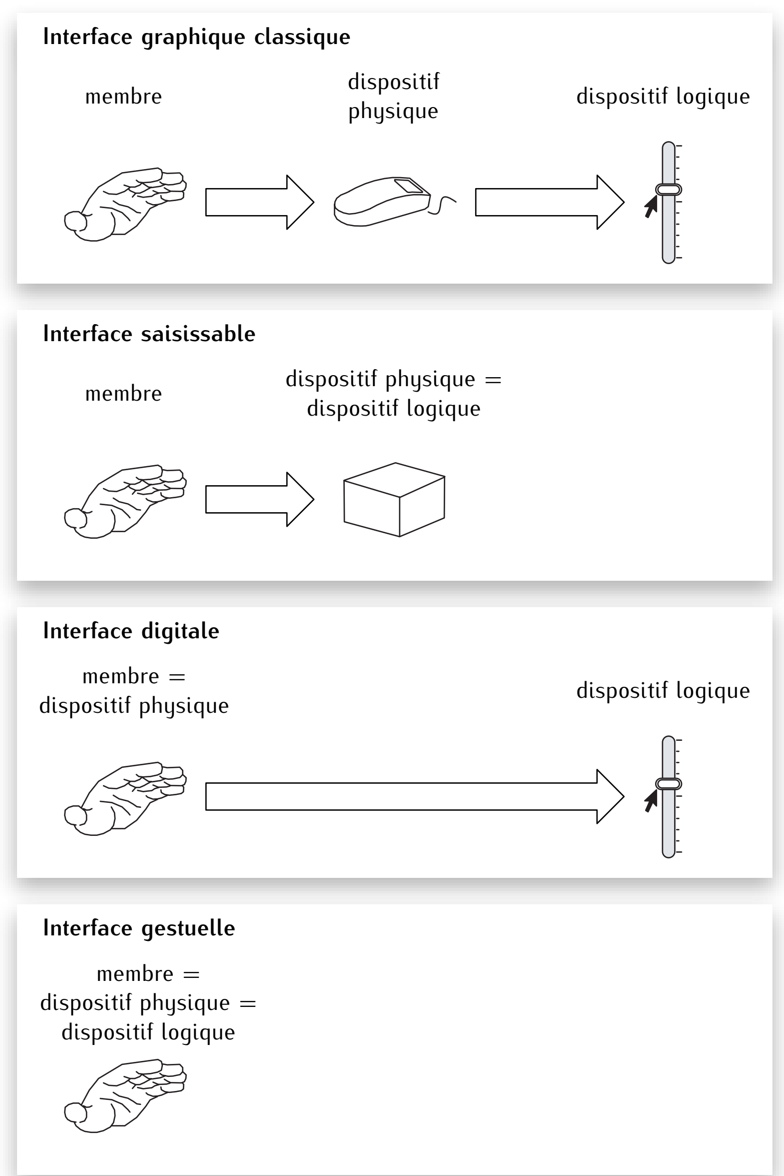


Figure 2.1 . Transition de l'interaction indirecte des interfaces graphiques classiques à différents paradigmes de manipulation directe.

D'après [Bérard, 1999b].
(a) Traditionnellement, un utilisateur manipule un périphérique de pointage, qui est couplé à un pointeur affiché, qui manipule un objet virtuel : la manipulation dite directe est en fait indirecte. L'immersion et la performance ne sont possibles qu'avec l'apprentissage au niveau moteur du gain du dispositif, et le lieu de l'action et celui du retour d'information restent séparés.
(b) L'interaction saisissable ou tangible est une forme d'interaction plus directe : le dispositif physique est identifié au dispositif logique. Des propriétés d'un objet physique sont couplés à des propriétés d'un objet virtuel — typiquement, leurs positions dans un plan.
(c) L'interaction digitale est une autre forme plus directe : le membre de l'utilisateur est utilisé comme dispositif d'entrée. Cette fois, c'est un membre de l'utilisateur qui est directement couplé à un objet physique.
(d) Enfin, l'interaction gestuelle identifie le membre de l'utilisateur, le dispositif physique, et le dispositif logique.

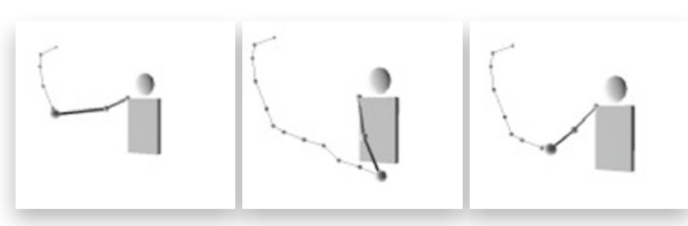


Figure 2.2 . Le *3D Blackboard*, un outil de dessin en trois dimensions.

D'après [Wu et al., 2000]. Une caméra observe l'utilisateur et permet d'évaluer la position et l'orientation de son bras et de son avant-bras (à gauche). Sa main joue alors le rôle de la craie, lui permettant de « desiner » dans l'espaces des formes qui peuvent être reproduites sur un écran (à droite).

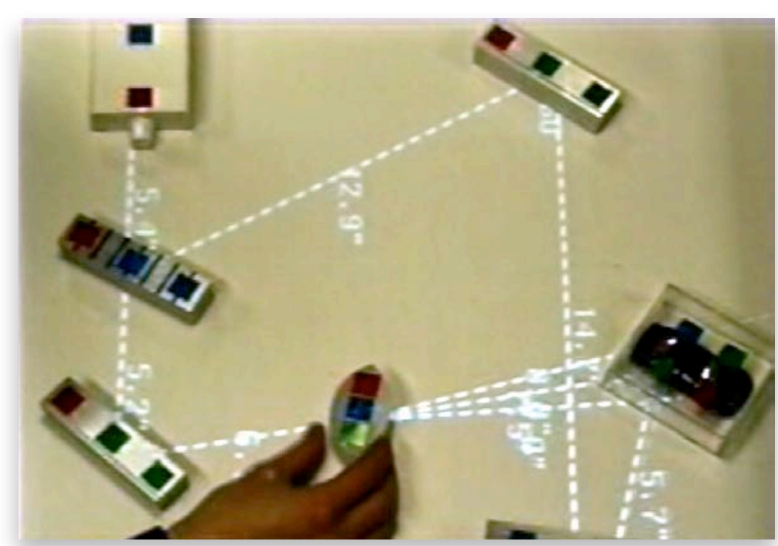


Figure 2.3 . Le projet *Illuminating Light*, une interface tangible pour la simulation d'un système optique.

Un montage d'optique complexe est simulé sur une surface augmentée. Les agents optiques (miroirs, lentilles) sont des objets réels ; les rayons lumineux virtuels sont affichés sur la surface par un projecteur vidéo. Cette simulation est rendue possible par la détection, *via* une camera, des objets tangibles posés sur la table. La tâche « réelle » correspondante à celle implémentée par ce système est le prototypage d'un système optique, sur un banc d'optique par exemple, en utilisant de vrais lentilles et miroirs.
Chaque objet « actif » présent sur la table est étiqueté d'un code couleur. Le logiciel perceptif repère ces codes pour déterminer l'identité, la position, et l'orientation des objets. Ces données sont fournies au logiciel de simulation, qui détermine les parcours théorique de la lumière dans le modèle optique, et projette une représentation des rayons lumineux et des attributs associés (angles, longueur des parcours).

la vision est le également mise en oeuvre dans le prototype qui a permis l'émergence du paradigme de *phicons* (abréviation de *physical icons*, ou icônes matérielles) [Ullmer et Ishii, 1997].

EnhancedMovie [Ishii et al., 2003] est une application de montage video pour bureau augmenté. Son interface est mixte : elle propose des interactions digitales et des interactions gestuelles. La figure 2.4 ci-contre résume l'évolution de la tâche de montage video, depuis la manipulation dans le monde physique à une manipulation de plus en plus directe dans le monde numérique.

Ces deux exemples montrent que la vision par ordinateur permet de réaliser des interfaces tangibles, digitales, et gestuelles réussies.

2.1.3 Un support des interactions de nouvelle génération

Nous venons de voir que la vision est particulièrement bien adaptée à la réalisation de surfaces interactives. Ceci peut en fait être généralisé à de nombreux besoins des systèmes interactifs de nouvelle génération.

L'informatique conventionnelle et les moyens de perception correspondants peuvent être résumés en quelques mots : un utilisateur seul travaille sur des informations numériques en utilisant un ordinateur (clavier, souris, écran), dans un bureau. Chacun de ces points peut être perçu comme une limitation pour l'utilisateur, liée à celles de la technologie correspondante. De fait, les différentes pistes d'exploration de l'informatique dite de nouvelle génération, et les domaines de recherches correspondants (en particulier, l'informatique ubiquitaire et l'informatique évanescente), s'inscrivent dans la prolongation d'au moins un de ces axes.

Nous évoquons maintenant différentes situations liées au mouvement de l'informatique ubiquitaire et dans lesquelles la vision par ordinateur semble pouvoir apporter des solutions aux problèmes d'interaction soulevés.

Un utilisateur devient un groupe d'utilisateurs. Ceci introduit de nouvelles caractéristiques. Le groupe peut être dispersé dans l'espace, auquel cas un lieu virtuel de l'interaction doit être construit par l'application. La vision permet par exemple d'observer des surfaces interactives distantes et d'afficher une image fusionnée sur chacune [Takao et al., 2003].

Un ordinateur unique dans un bureau devient un ensemble d'ordinateurs répartis dans un environnement : bureau ou maison équipée par exemple. C'est le domaine de l'informatique ambiante, où un utilisateur interagit avec de nombreux appareils. Ces ordinateurs peuvent éventuellement être mobiles (informatique ubiquitaire), voire devenir invisibles pour l'utilisateur, et l'interaction avec eux implicite (informatique évanescente). La vision permet cette forme d'interaction « à distance » — sans instrumenter les utilisateurs.

Le triplet *clavier, souris, écran* disparaît et laisse la place à d'autres outils d'interaction. Il peut s'agir d'une surface d'affichage tactile. La manipulation des objets virtuels se fait directement avec les doigts et les mains, sans intermédiaires (c'est l'interaction digitale). Il peut également s'agir, par exemple, d'une interface de pilotage par gestes du système, ou d'une interface de grande taille. Nous verrons dans les chapitres suivants que la vision est particulièrement bien adaptés à la réalisation de surfaces tactiles.

Au lieu d'*informations numériques*, il est possible de manipuler des objets physiques (interaction tangible) qui agissent sur le monde numérique (virtualité augmentée) ou qui sont améliorés par de l'information numérique (réalité augmentée). Pour d'autres tâches, l'affichage peut se faire sur des objets anodins qui réagissent au comportement de l'utilisateur et aux évolutions de l'environnement (interaction calme ou périphérique). L'exemple d'*Illuminating Light*, présenté plus haut dans ce chapitre, montre le rôle de la vision pour de tels systèmes.

Figure 2.4 . Un exemple de transition de la manipulation directe dans le monde physique vers la manipulation directe dans le monde numérique : le montage video.

(*a*, *b*) Manipulation dans le monde physique à l'aide d'outils mécaniques. Massicot à bandes *Smith splicer*, adhésif *3M Scotch Brand*. [Calaway, 2006].
(*c*, *d*) Manipulation indirecte dans le monde physique. à l'aide de l'électronique, l'édition linéaire est effectuée en construisant manuellement des *edit decision lists* (EDL) qui sont ensuite compilées par une machine pour produire le montage. À gauche, Ampex EDM-1, 1976 ; à droite, Sony BVE-910, 1981.
(*e*, *f*) Transposition dans le monde numérique : édition video virtuelle, manipulation « directe » dans le monde numérique. À gauche, le prototype de [Mackay et Davenport, 1989] ; à droite, *iMovie*, Apple Computer, 2002.
(*g*, *h*) Édition video en virtualité augmentée, manipulation plus directe (digitale et gestuelle) dans le monde numérique, d'après [Ishii et al., 2003]. Une représentation d'un chûtier est projetée sur la surface, et un utilisateur peut manipuler les chutes avec des gestes naturels (attraper, déplacer, couper) et des gestes supplémentaires pour les propriétés purement numériques des objets (jouer la video, annuler, etc.)

Enfin, le *bureau* lui-même peut être remplacé par un environnement contraignant. L'environnement de l'interaction peut ne pas être sûr, en termes de confidentialité par exemple ; une borne interactive dans un abribus doit répondre à des contraintes différentes de celles d'un ordinateur de bureau. Les conditions peuvent être difficiles pour le matériel — c'est le cas, par exemple, pour un système interactif utilisé sur un terrain militaire, en extérieur, ou de nuit, dans des conditions météorologiques défavorables. La vision permet le déport de tout le matériel (ordinateur, caméra, eventuellement video-projecteur) loin du lieu de l'interaction, permettant de résoudre ces problèmes.

2.1.4 Conclusion

La vision par ordinateur permet la réalisation de nombreuses formes d'interaction innovantes spécalisée pour la réalisation de tâches particulières. En cela elle apporte le même bénéfice que les dispositifs d'interaction spécialisés : une interaction mieux adatée à la tâche que celle offerte par la souris. L'atout particulier de la vision est qu'elle permet la création de nombreuses interactions différentes au travers d'un même dispositif physique : la caméra. En particulier, la vision permet la perception des gestes de l'utilisateur sans nécessairement instrumenter ces membres. Elle permet donc une interaction plus directe qu'à la souris ou au travers d'un quelconque dispositif d'interaction. Enfin, le champ d'action de la vision étant relativement plus étendu que celui des dispositfs classique, il permet d'envisager l'extension du lieu d'interaction à un espace plus vaste que la station de travail standard, l'interaction avec plusieurs utilisateurs, et la mobilité de ces utilisateurs. Ces points constituent des atouts essentiels pour l'informatique ubiquitaire et évanescente.

La vision par ordinateur est donc bien adaptée à la résolution de certains nouveaux problèmes d'interaction. Il existe d'ailleurs des prototypes de systèmes interactifs performants l'utilisant. Il est donc surprenant que son utilisation soit si peu répandue. C'est le paradoxe que nous étudions maintenant.

2.2 Pourquoi la vision est elle sous-utilisée ?

Pour l'industrie comme pour la recherche en interaction, l'utilisation de la vision pour l'interaction est anecdotique. À notre connaissance, les seuls systèmes interactifs en production qui exploitent la vision sont l'*eyeToy* de Sony et la *Wiimote* de Nintendo (figure 2.5 ci-contre), deux accessoires de consoles jeux video.

Comment expliquer cette sous-utilisation de la vision alors qu'elle semble offrir un potentiel important ? Nous détaillons ici nos hypothèses. La sous-utilisation de la vision est liée à deux manques : certains services de vision par ordinateur nécessaire à la réalisation des systèmes interactifs n'ont simplement encore jamais été conçus. C'est le défaut d'utilité de la vision. Mais même lorsque les services dont disponibles, il le sont souvent sous une forme qu'il est difficile de mettre en oeuvre de la part des développeurs de systèmes interactifs. C'est le défault d'utilisabilité.

2.2.1 Deux cas de non-utilisation de la vision

Nous présentons ici deux exemples de systèmes interactifs dont les concepteurs ont fait le choix de ne pas utiliser la vision par ordinateur alors qu'il aurait pu être pertinent. Ces exemples font respectivement ressortir le défaut d'utilité et le défaut d'utilisabilité de la vision, que nous développons dans les sections suivantes.

2.2.1.1 Interaction digitale

Quand l'interaction est restreinte à une seule modalité, le choix effectué est souvent d'utiliser des surfaces équipées, plus faciles à déployer car prêtes à l'emploi.

C'est le cas, par exemple, lorsque les techniques d'interaction digitale avec des objets virtuels sont explorés. Une telle étude est *Release, Relocate, Reorient, Resize* [Ringel et al., 2004] : il s'agit d'explorer des techniques d'interaction pour le partage de documents sur une table multi-utilisateur. Les auteur n'ont pas eu d'effort à fournir pour débuter l'expérimentation avec la table *DiamondTouch*, car c'est un système robuste et stable.

Au contraire, l'intégration et le déploiement de l'application de dessin collaboratif *DoodleDraw* (représentée fig. 2.6 page suivante, et que nous décrivons page 153) en utilisant la vision nous a demandé plusieurs années-homme de travail. En particulier, nous avons implémenté un outil de détection et suivi des doigts qui permet de rendre une surface interactive [Borkowski et al., 2006]. Ceci aurait été prohibitif pour une équipe de recherche n'étant pas concernée par les problématiques de vision par ordinateur et d'architecture logicielle liées.

2.2.1.2 Interaction gestuelle

L'intérêt de la vision est plus général que pour les seules surfaces interactives, et choisir la vision par ordinateur comme support de l'interaction peut être valide dans d'autres cas. C'est vrai, par exemple, pour l'interaction avec un espace tridimensionnel.

Pour des tâches d'interaction en trois dimensions il est possible d'utiliser le *3D Blackboard* présentée dans [Wu et al., 2000], qui est fondé sur des techniques de vision artificielle monoscopique. Cependant ce type de prototype demande un apprentissage long, est difficile à déployer, et fournit des performances (latence, précision, et robustesse) trop basses.

Là encore, le choix de la vision ne sera généralement pas retenu pour la réalisation d'un système interactif : par exemple, [Vogel et Balakrishnan, 2005] présente une étude de différentes techniques d'interaction à distance. Pour réaliser un prototype les auteurs ont exploré la piste de la vision mais ont finalement choisi d'utiliser Vicon

Figure 2.5 . La caméra *EyeToy* et la *Wiimote*.

À gauche : la caméra *EyeToy* de Sony (en haut), une extension de la console de jeux *PlayStation 2* de Sony. Associée à un détecteur de mouvement et de couleur, elle permet la création d'interfaces gestuelles primitives (en bas).
À droite : la *Wiimote* (en haut), le contrôleur de jeu de la console *Wii* de Nintendo. Cette manette embarque une caméra infrarouge, qui observe un émetteur placé au-dessus de l'écran. En bas, la vue de la caméra, observée *via* le dialogue de calibrage de la *Wiimote*. Elle permet la construction d'interfaces plus élaborées.

(www.vicon.com) : un dispositif matériel spécifiquement dédié au suivi d'utilisateurs équipés et utilisé principalement pour la capture de mouvement par l'industrie du cinéma.

2.2.2 Défauts d'utilité

La vision par ordinateur est un problème de recherche ouvert. Comme l'exprime Matthiew Turk (dans la citation introductive de ce chapitre), la vision dans son ensemble est un problème qui est loin d'être résolu. Le nombre de problèmes de vision par ordinateur qui sont maîtrisé est encore très limité. Il serait vain de tenter ici d'expliquer les difficultés fondamentales des problèmes de vision car ce serait déjà être sur la voie de solutions. Nous constatons simplement qu'aucune solution de vision par ordinateur n'a été proposée pour de nombreux problèmes soulevés par des besoins en interaction Homme-machine.

Il faut toutefois nuancer ce constat. Par exemple : l'identification visuelle d'utilisateurs

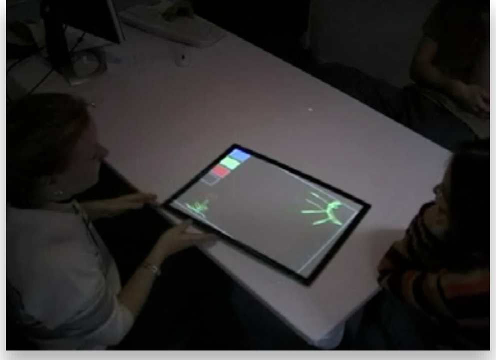


Figure 2.6 . *DoodleDraw*, un système de dessin collaboratif sur surface augmentée fixe, portable, ou dirigeable.

Figure 2.7 . *Interactive Public Ambient Displays*

IPAD est une interface « ambiante » qui permet la transition de l'interaction implicite et l'affichage périphérique vers l'interaction fortement couplée et la manipulation directe.
Au repos, l'écran affiche des informations générales, pertinentes pour tout observateur potentiel (heure, prévisions météorologiques). Lorsque des utilisateurs arrivent en périphérie du dispositif, ils sont identifiés, et des informations personnalisées mais non privées sont affichées. Enfin, lorsque des utilisateurs parviennent à proximité, l'interaction directe devient possible, par des gestes de pointage (à distance) et en touchant la surface tactile.
La localisation des utilisateurs et la détermination de leur pose (ie. s'ils regardent le dispositif ou non) est effectuée à l'aide d'un système commercial de suivi de marqueurs en infrarouge, utilisé par exemple pour la capture de mouvement dans l'industrie cinématographique.
À notre connaissance, l'identification est simulée par un « magicien d'Oz ».

est un problème pertinent pour l'interaction, i.e. qui a de nombreuses applications en interaction Homme-machine. Il est exploité dans le système *Interactive Public Ambient Displays* [Vogel et Balakrishnan, 2004] (figure 2.7 ci-contre), pour afficher des informations personnalisées ou privées sur un écran public, selon l'identité du ou des utilisateurs présents. Comme d'autres problèmes, l'identification visuelle largement traité en vision par ordinateur et elle est considérée comme résolue dans certains contextes. Les meilleurs systèmes existants permettent une identification fiable à 70% selon les états de l'art de [Senior et al., 2002] et [Zhao et al., 2003].

Mais les contraintes imposées par les systèmes de vision par ordinateur (un unique utilisateur est présent dans le champ de la caméra, il est vu de face, de près, avec un éclairage contrôlé) sont incompatibles avec les contextes d'usage des systèmes interactifs envisagés. Par ailleurs, l'interaction impose elle aussi certaines contraintes (faible temps de réponse, précision) que les systèmes de vision ne prennent pas en compte. Finalement, même si le problème semble traité et bien maîtrisé dans la communauté de vision par ordinateur, il n'existe pas de système de vision prêt à fournir un service utile aux systèmes interactifs.

2.2.3 Défauts d'utilisabilité

Comme nous venons de le constater, la vision ne fournit pas toujours les « bons » services pour l'interaction. Mais même lorsque des résultats publiés montrent qu'elle peut satisfaire un besoin particulier, elle n'est pas retenue : le coût d'apprentissage et de développement pour déployer un service fondé sur la vision est supérieur au seuil accepté par les clients potentiels. C'est ce que nous montrons ici.

2.2.3.1 Un existant mal ciblé

D'après notre expérience, la conception et la réalisation d'un système perceptif est une opération où l'existant n'est peu ou pas réutilisé. Un cas d'étude semble pertinent : la *Table Magique* [Bérard, 2003] développée dans notre équipe de recherche au laboratoire CLIPS-IMAG (figure 3.9 page 48). L'un des objectifs de ce prototype est de donner vie aux interactions digitales avec des documents virtuels, proposées dans le projet *DigitalDesk* dix ans plus tôt [Wellner, 1993b].

Les techniques de vision par ordinateur mises en jeu pour la réalisation de la *Table Magique* sont bien établies. Il s'agit, en particulier, de la détection et du suivi d'objets colorés, du calibrage géométrique automatique entre surfaces bidimensionnelles, et du seuillage adaptatif. Certaines de ces techniques ont d'ailleurs été décrites pour le *DigitalDesk* : le seuillage adaptatif [Wellner, 1993a] et le calibrage automatique [Wellner, 1993c]. Or le développement de la *Table Magique* a imposé le développement de chacune de ces techniques de vision par ordinateur. L'expérience ecquise lors de son développement nous a montré qu'aucun existant satisfaisant ne pouvait fournir les fonctionnalités requises, sous les contraintes de l'interaction — c'est-à-dire, avec des performances suffisantes.

Cette tendance est générale. Par exemple les systèmes *Activity from posture* [Jaimes, 2006] ou *TouchLight* [Wilson, 2004], que nous décrirons plus loin dans ce document, sont conçus *ex nihilo* en ce qui concerne la vision par ordinateur.

2.2.3.2 Un effort de factorisation balbutiant

À l'opposé de cette tendance, il existe des tentatives pour mettre en commun les résultats sous forme de bibliothèque logicielle. La plus connue, et la plus répandue, est *OpenCV* [Bradski et Pisarevsky, 2000], une bibliothèque de fonctions de vision temps réel pour l'interaction Homme-machine fournie par Intel Research. Elle a été utilisée pour fournir des algorithmes dans la construction de nombreux systèmes, par exemple *Gesture + Play* [Konrad et al., 2003] ou *The Designer's Outpost* [Klemmer et al., 2001]. Cette bibliothèque fournit un support pour l'acquisition multi

plate-formes d'un flux video, un certain nombre de primitives de traitement d'image, et des techniques de vision avancées comme la détection et la reconnaissance de visages, la détection d'objets, ou la détection de mouvement.

Supposons qu'un chercheur souhaite implémenter un système interactif minimal qui affiche les visages détectés par une caméra. Il s'agit de l'une des fonctionnalités de base de la bibliothèque *OpenCV*, et constitue d'ailleurs l'un des exemples de démonstration. Pour cette tâche, l'utilisateur doit écrire plusieurs centaines de lignes de code C++, après avoir compris sommairement ce que sont et comment fonctionnent les *Haar Like Features* — une méthode complexe pour modéliser l'apparence d'une classe d'objets, fondée sur une décomposition de l'image en ondelettes de Haar. Ceci n'est pas acceptable pour un développeur de systèmes interactifs, qui est rarement un spécialiste en vision par ordinateur.

Citons un second exemple. D'après son auteur, la fonctionnalité perceptive du *Designer's Outpost* (la détection et la localisation de Post-It sur un tableau), utilisant *OpenCV*, a nécessité environ 3000 lignes de code [Klemmer et al., 2004]. D'après notre expérience, le gain en temps et en volume de développement apporté par cette bibliothèque n'est pas interessant par rapport à un développement en partant de zéro qui aurait nécessité un effort similaire.

Il semble que l'effort de factorisation de services de vision ait lieu en partant des bas niveaux d'abstraction, c'est-à-dire qu'il est orienté par les fournisseurs de services, experts en vision par ordinateur, au lieu d'être guidé par les besoins des utilisateurs — les développeurs de systèmes interactifs.

2.2.4 Conclusions

Schackel classe les facteurs humains pertinents pour une interface homme-machine selon trois axes [Shackel, 1991] :

- utilité : le système fait-il ce qui est requis fonctionnellement ?
- utilisabilité : les utilisateurs l'utiliseront-ils avec succès ?
- appréciabilité : les utilisateurs le jugeront-ils acceptable ?

La norme ISO 9241 (*Guidance on Usability*) définit l'utilisabilité comme « le degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction, dans un contexte d'utilisation spécifié ». [Nielsen, 1994] décompose également le concept d'utilisabilité en cinq caractéristiques majeures d'un système utilisable. L'efficience (*efficient to use*) et la satisfaction (*subjective satisfaction*) se retrouvent telles quelle dans la norme ISO 9241. La facilité d'apprentissage (*easy to learn*), la facilité d'appropriation (*easy to remember*) et la fiabilité (*few errors*) peuvent être considérées comme des composantes de l'efficacité.

Nous reprenons ces critères dans une situation légèrement différente de celle pour laquelle ils ont été émis. Pour nous, l'utilisateur est un développeur de système interactif, non pas un utilisateur final. Notre utilisateur vise à utiliser un système qui fournit des services de vision par ordinateur.

D'après ce que nous avons présenté dans cette section, nous formulons le postulat suivant :

> Si la vision par ordinateur est sous-utilisée dans le contexte de l'interaction Homme-machine, c'est parce que, en général, elle n'est pas utile ou utilisable.

En effet, certains besoins fonctionnels correspondent à des problèmes non résolus en vision. D'autres ne sont pas fournis dans des bibliothèques logicielles. D'autres sont fournis mais ne le sont pas de manière satisfaisante (ou utilisable) : soit les performances d'un service particulier sont en-deça du besoin des développeurs, soit son utilisation est trop complexe.

Malgré l'intérêt de la vision pour l'interaction, il n'est pas surprenant que les concepteurs de systèmes interactifs fassent souvent le choix de ne pas l'utiliser.

2.3 Objectifs de la thèse

Notre objectif est de développer l'usage de la vision par ordinateur dans la réalisation des interactions Homme-machine. Notre probléme est de rendre la vision utile et utilisable pour les développeurs de systèmes interactifs afin de contribuer à la démocratisation de son usage en interaction Homme-machine.

Notre plan de travail conciste à déterminer les requis d'utilité et d'utilisabité d'une « bonne » boîte à outils de vision, puis de proposer une approche de conception logicielle adaptée, de l'implémenter, et de l'évaluer.

2.3.1 Approche conceptuelle

Nous adoptons un approche en deux parties : sur le fond et l'utilité d'une part, et sur la forme et l'utilisabilité d'autre part. Il s'agit, pour le premier point, de déterminer le contenu de notre future boîte à outils, et pour le second, de déterminer quelle interface elle devra fournir à ses clients. En d'autres termes, nous identifions les **requis** que nous devons satisfaire, et qui peuvent être définis ainsi :

> « Les *requis fonctionnels* établissent la fonctionnalité des composants [logiciels] dans le domaine d'application ; les *requis non fonctionnels*, comme la précision, la sécurité, l'utilisabilité, le coût, ou la performance, et autres placent des contraintes globales sur la manière dont la fonctionnalité est présentée. » [Chung, 1991]

Nous commencerons par étudier un ensemble de systèmes interactifs existants qui ont fait l'objet de publications scientifiques. Notre objectif est de déterminer l'ensemble des services de perception artificielle qu'ils utilisent ou pourraient utiliser pour permettre au système de percevoir l'utilisateur et son environnement. Nous proposons une taxonomie descriptive de ces services, qui nous amène à un ensemble recouvrant (ou suffisant) de services à réaliser. Ces points font l'objet des sections 3.3.2 page 58 et 5.1.2 page 90, et constituent la démarche permettant de satisfaire les critères d'utilité et d'utilisabilité sur les services offerts par la boîte à outils.

Afin de fournir une boîte à outils elle-même utilisable, nous étudions en parallèle les approches existantes en termes de bibliothèques logicielles, en particulier de vision par ordinateur. Nous dégageons un ensemble de requis fonctionnels pour son architecture ainsi qu'un ensemble de requis non fonctionnels (en particulier en termes de performance système et de performance utilisateur). Ceci fait l'object de la section 4.3 page 79. Cette étude nous amène à proposer une architecture répondant à ces requis au chapitre 5 page 87.

2.3.2 Mise en oeuvre, validation, et évaluation

La réalisation de la boîte à outils a un objectif double :

- la validation de l'approche ;
- la fourniture de services de vision par ordinateur aux concepteurs d'interactions Homme-machine.

Pour valider l'approche, il conviendra d'implémenter l'architecture (la « coquille » de la boîte à outils), et suffisamment de services pour pouvoir reproduire un ou plusieurs systèmes interactifs décrits dans l'état de l'art du chapitre 3. Pour permettre la fourniture de services de perception visuelle, il faudra en outre démontrer qu'il est possible d'encapsuler les résultats techniques des chercheurs en vision par ordinateur dans notre boîte à outils, les rendant ainsi utilisables.

L'évaluation de la boîte à outils comprendra également deux parties : d'une part, il sera nécessaire de vérifier que les services fournis sont utiles (ils permettent de construire les systèmes étudiés) et utilisables (ils satisfont à des critères de performance déterminés). Enfin, l'adéquation aux besoins de la boîte à outils sera jugée à travers sa facilité de diffusion au sein d'une équipe de recherche.

3 Requis fonctionnels pour une boîte à outils utile et utilisable

> *« Why is programming so difficult ? Part of the problem is that it requires problem solving skills and great precision, but this does not fully explain the difficulty. Even when a person can envision a viable detailed solution to a programming problem, it is often very hard to express the solution correctly in the form required by the computer. This is a user-interface problem that has long been recognized but neglected. »*
>
> *[Pane, 2002]*

Une boîte à outils est un logiciel dont l'interface utilisateur est une API (*Application Programmer Interface*), et dont les utilisateurs sont des programmeurs [Klemmer et al., 2004]. Il est donc possible d'adopter une approche centrée utilisateur pour la concevoir. En d'autres termes, s'appuyer sur une analyse des besoins des utilisateurs permet d'aboutir à une boîte à outils qui sera adaptée à leurs besoins.

Dans notre cas il est possible d'identifier quatre catégories d'utilisateurs distinctes ayant des besoins différents et qui ne sont pas nécessairement compatibles. Pourtant, pour réussir, une boîte à outils devra autant que possible les satisfaire. Au long de ce chapitre et du suivant, nous désignerons ces classes par des prénoms afin d'y référer plus facilement. Les lecteurs familiers avec les équipes PRIMA et IIHM pourront peut-être les reconnaître.

Commençons par donner une typologie de ces utilisateurs :

Caroline est un utilisateur final (*end user*). Elle exécute des tâches en utilisant un système interactif dont la partie perceptive utilise notre boîte à outils. Elle n'a aucune connaissance en vision par ordinateur ni en développement d'interactions Homme-machine.

Laurence est concepteur et développeur d'un système interactif. Elle utilise la boîte à outils pour fournir une entrée perceptive à son système, et implémente des techniques d'interaction à partir de ces entrées. Elle est experte en interaction Homme-machine et compétente en génie logiciel mais n'a pas d'expérience de vision par ordinateur.

Patrick est intégrateur de services. Il encapsule des algorithmes de vision de la boîte à outils pour en faire des services, assemble des services existants, ou les adapte. Il est expert en génie logiciel et est familier des domaines de la vision par ordinateur et de l'interaction Homme-machine.

Stanislas est expert en vision par ordinateur. Il conçoit et implémente des algorithmes de perception. Il est expert en vision par ordinateur et en traitement d'image, mais n'a pas de compétence dans les autres domaines.

Concrètement, en transposant les rôles de ces utilisateurs dans les exemples présentés au cours de ce document : Stanislas produit des parties de *OpenCV* ou *ARToolkit*, Patrick fournit *EasyLiving*, Laurence produit l'application *Caretta*.

Les relations entre chacun de ces utilisateurs et la boîte à outils sont résumés sur la figure 3.1. Dans le cadre de cette thèse, nous adoptons les points de vue de Patrick et de Stanislas pour implémenter la boîte à outils *gmlVision* (ce sera l'objet du chapitre 6 page 117), puis celui de Laurence pour réaliser des systèmes interactifs qui l'utilisent (chapitre 7 page 143).

Structure du chapitre.

Dans ce chapitre, nous nous interessons aux requis imposés par Laurence et Caroline. Nous devons déterminer quels services perceptifs seront utiles à Laurence pour bâtir un système interactif, et quelles contraintes doivent être appliquées à ces services pour qu'ils soient utilisables par Caroline. En d'autres termes, nous répondons ici à la question : quels services perceptuels fournir ?

Après avoir introduit la famille des systèmes interactifs qui nous interessent, nous commençons par montrer que le choix de la vision pour implémenter la perception n'est pas une évidence (section 3.1), et qu'il résulte de son **utilité** et de celle des outils logiciels correspondants. Nous supposons par la suite que son utilisation a été retenue par le concepteur de système interactif et nous ne cherchons donc pas à la justifier.

Nous présentons ensuite un état de l'art structuré des systèmes interactifs de nouvelle génération existants où la vison a, ou pourrait avoir, un intérêt (section 3.2). Nous utilisons cet état de l'art pour identifier les services perceptifs requis par ces systèmes : cette première section est un analyse fonctionnelle des besoins de Laurence.

Enfin, nous synthétisons ces résultats dans une dernière partie (section 3.3.2), ce qui nous amène à proposer un ensemble de requis sur le « fond », ou le contenu, d'une boîte à outils adaptée. Nous proposons une taxonomie qui permet d'exprimer des requis de manière plus formelle, tout en restant intelligible pour nos quatre classes d'utilisateurs.

3.1 Préambule : Vision dans les systèmes interactifs

Afin de clarifier le choix des systèmes que nous retenons dans notre état de l'art, nous décrivons tout d'abord notre cadre scientifique. Dans ce préambule, nous présentons brièvement une famille des systèmes interactifs dits *post-WIMP* pour lesquels

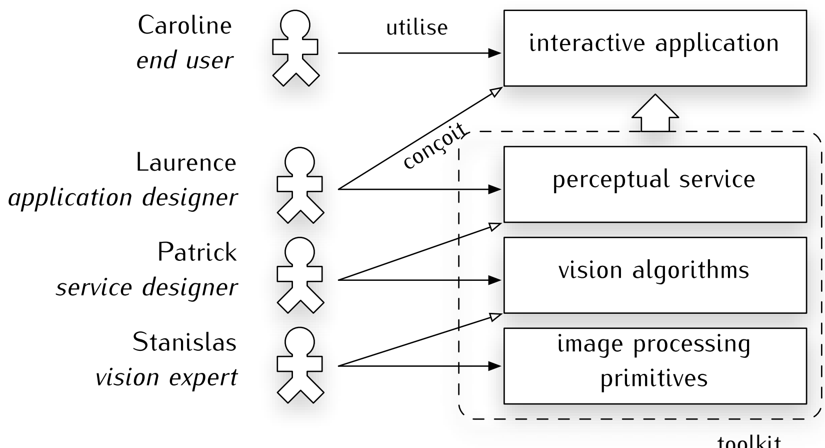


Figure 3.1 . Relations entre les quatre catégories d'utilisateurs et les différentes facettes de l'interface homme-machine de la boîte à outils *gmlVision*.

la vision a un intérêt. Leurs concepteurs font le choix d'utiliser la vision lorsque l'implémentation du service perceptif correspondant est triviale ou déjà disponible — réciproquement, ils utilisent d'autres technologies lorsque elles sont plus accessibles (plus utiles ou utilisables). Nous illustrons ces points à partir de quelques exemples de systèmes.

Notre objectif n'est pas de convaincre Laurence d'utiliser la vision. Nous supposerons en fait durant l'état de l'art que **l'utilisation de la vision a été retenue** par le concepteur d'applications interactives. Nous nous contentons de déterminer comment la rendre utile et utilisable.

3.1.1 Les systèmes interactifs post-WIMP

L'une des premières apparitions du terme WIMP — pour *windows, icons, mouse, and pointer*, ou fenêtres, icônes, souris, et pointeur — est publiée dans [Edwards, 1988]. Il est à l'époque utilisé pour qualifier la famille d'interfaces homme-machine graphiques qui est depuis un standard de fait pour la conception et le développement d'applications : seuls la souris et le clavier sont utilisés comme périphériques d'entrée ; ils contrôlent un pointeur souris ou un curseur d'édition de texte ; enfin, les entités numériques sont éventuellement représentées par des icônes (manipulables avec le pointeur) et regroupées dans des fenêtres.

Lorsqu'un concepteur de système interactif — en particulier, les chercheurs en interaction Homme-machine — décide de créer un système innovant en s'affranchissant des limites du consensus WIMP, il est convenu de parler de système *post-WIMP*. Les systèmes post-WIMP qui nous intéressent sont ceux où l'action de l'utilisateur n'est plus effectuée et perçue par le biais du clavier et de la souris, mais par celui d'autres technologies.

Ainsi que nous l'avons évoqué dans le chapitre précédent, 2.1.3 page 26, cette catégorie d'innovations ouvre de nombreuses possibilités pour réaliser des sytèmes interactifs utiles ou des techniques d'interaction plus performantes. Mais elle fait également apparaître de nouveaux besoins en termes de perception, besoins pour lesquels il n'existe pas toujours de solution.

Nous structurerons notre état de l'art des systèmes interactifs suivant un ensemble de catégories de nouveaux besoins de perception. Chaque catégorie correspond à un axe de recherche différent dans le domaine de l'interaction Homme-machine.

Le sous-ensemble des systèmes post-WIMP fondé sur l'utilisation d'une surface interactive fait partie des systèmes d'intérêt. Pour rendre une surface d'affichage interactive, de nombreuses solutions alternatives à l'utilisation de la vision existent. Certains utilisent, pour rendre une surface tactile, les variations de propriétés optiques de la matière lors d'un contact (figure 3.3 page 39) ; d'autres la conductivité du corps humain (figure 7.17 page 167).

Il est particulièrement intéressant d'étudier ces systèmes pour concevoir une solution fondée sur la vision ayant des objectif équivalents. En effet, ces systèmes ont tous été développés pour des spécialistes en interaction Homme-machine. L'objectif est en général d'explorer de nouvelles techniques d'interaction, en particulier digitales : ces systèmes répondent donc à des besoins que nous devons également satisfaire dans *gmlVision*. Nous pourrons également comparer nos résultats avec ces dispositifs, ce qui sera fait dans le chaptitre 7 page 143, *Validation et Évaluation*.

3.1.2 Faire le choix d'utiliser la vision

Au cours du chapitre précédent, nous avons eu pour objectif d'illustrer l'intérêt pour la vision par ordinateur, afin de faire partager au lecteur notre conviction que son utilisation peut être pertinente pour l'interaction.

Ici, nous voulons montrer que l'utilisation de la vision et un **choix** parmi d'autres, et que la solution selectionnée et celle qui répond à des contraintes d'utilité et d'utilisabilité : ceci justifie et introduit notre analyse fonctionnelle, à laquelle la suite du chapitre sera consacrée.

La vision pour la perception n'est pas un choix binaire : il existe en fait une gradation. À une extrémité, certains périphériques physiques exploitent la vision par ordinateur pour fournir la perception, mais sont fonctionnellement semblables à un périphérique électronique ordinaire comme une souris ou un écran : il n'apportent pas tous les avantages que peut fournir la vision. Par exemple, le *DViT* (figure 3.2) est un écran augmenté par la vision. Plusieurs caméras linéaires observent le voisinage immédiat de la surface de l'écran, en coupe, afin de détecter le contact d'éventuels objets. Face aux caméras, les bords de l'écran sont équipés de LEDs infrarouge pour faciliter la détection. Les données des caméras sont traitées sur un ordinateur intégré à l'écran. L'écran devient ainsi tactile. À l'autre extrémité de la gradation, il est possible de qualifier certains systèmes perceptifs de « purement visuels ». Le *3D Blackboard* [Wu et al., 2000] présenté précedemment appartient à cette catégorie : le dispositif de perception est indépendant (et géographiquement éloigné) du lieu de l'interaction. La caméra est le seul dispositif mis en oeuvre dans la perception. En particulier, la surface d'interaction n'a besoin d'aucun équipement particulier.

Remarquons enfin que la vision n'est dans certains cas qu'une solution parmi d'autres pour réaliser des systèmes perceptifs. Prenons l'exemple du jeu *TROC* [Renevier, 2004], un logiciel collaboratif synchrone de réalité augmentée mobile, réalisé dans l'équipe de recherche de l'auteur. La tâche des joueurs de *TROC* est de récupérer, dans un conteneur virtuel qu'ils transportent, des objets virtuels placés dans le monde réel. Ils observent leur environnement à travers un casque semi-transparent, et les objets recherchés sont affichés dans le casque des joueurs à une place définie dans le monde réel. L'applicatif doit donc avoir connaissance à tout instant de la localisation des joueurs dans l'espace (c'est-à-dire leur position sur un plan du terrain de jeu, en deux dimensions) et de l'orientation de leur tête (afin d'afficher les objets à la bonne position dans leur champ visuel). Les concepteurs ont exploré diverses technologies (en particulier triangulation radio, accéléromètres, et vision) pour finale-

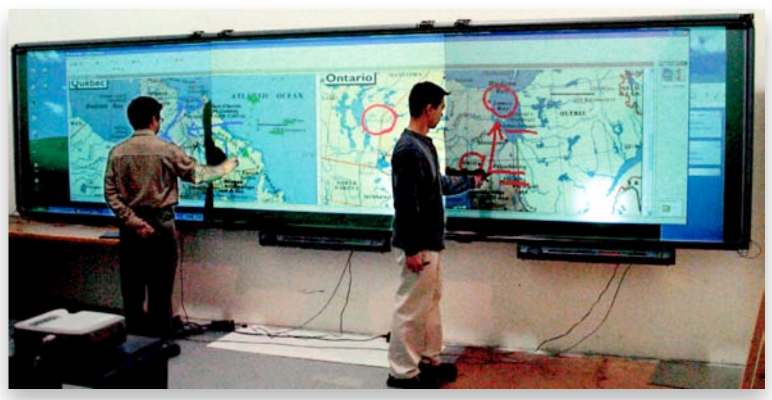


Figure 3.2 . Le tableau interactif *DViT* conçu par Smart Technologies

D'après [Morrison, 2005].
Plusieurs écrans plats sont ici équipés de quatre caméras monochrome quasi linéaires (ayant une résolution horizontale bien plus grande que leur résolution verticale) à chaque coin. Les caméras ont un angle de vue de 90° : chacune voit donc la totalité de région au voisinage de la surface, dans une épaisseur de quelques millimètres. Lorsqu'un doigt ou un stylo entre dans cette zone, chaque caméra perçoit une baisse de luminosité localisée. Par triangulation, le système détermine le lieu du contact ; cette méthode limite l'interaction à deux points de contact, interdisant la manipulation multi-utilisateurs à deux mains.

ment se contenter d'un « magicien d'Oz » (simulation d'un capteur par un opérateur humain) pour l'évaluation de l'interaction.

Dans ce cas particulier, la vision n'a pas été retenue faute de solution « prête à l'emploi. » Elle a néanmoins été explorée par les concepteurs. D'une part, c'est la solution qui demande le moins d'instrumentation de l'environnement. D'autre part, parmi les solutions explorées, c'est la seule qui permet de réaliser avec le même équipement les deux types de perception requises : localisation et orientation. La vision apparait ainsi souvent comme une technologie de perception non-invasive généraliste, qui permet de percevoir de nombreuses propriétés du monde avec un seul dispositif physique. Comme nous l'avons évoqué précédemment, en déportant la perception dans le cognitif, les dispositif de perception sont dématérialisés : la vision est effectivement une méthode de perception générale.

3.2 Des systèmes interactifs de nouvelle génération aux services perceptifs

Pour déterminer quels services perceptifs il convient de fournir à Laurence, nous proposons d'étudier des systèmes interactifs existants. Les systèmes pertinents pour notre travail sont ceux qui :

- exploitent la vision par ordinateur comme moyen de perception des actions de l'utilisateur ; ou
- n'utilisent pas la vision par ordinateur mais pourraient l'utiliser pour cette tâche, et dont la réalisation serait facilitée par l'emploi de la vision. Dans ce second cas, nous nous appuyons sur notre expertise pour déterminer la faisabilité d'un service perceptif fondé sur la vision par ordinateur.

Démarche adoptée.

Nous commençons ici un état de l'art des systèmes interactifs pour lesquels la vision a un intérêt. Comme expliqué au chapitre précédent (2.1.3 page 26), les axes d'innovation à l'origine de ces systèmes correspondent grossièrement à la négation de chacun des postulats du paradigme WIMP. Par conséquent, pour plus de lisibilité nous structurons l'étude de ces systèmes selon chacun de ces axes. Bien entendu, les

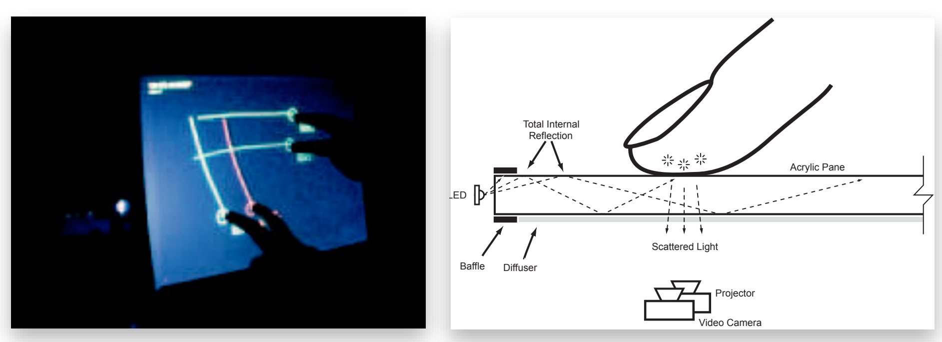


Figure 3.3 . Une table augmentée utilisant la réflexion totale frustrée.

D'après [Han, 2005].
Une surface de plexiglas transparente est dépolie du côté éloigné de l'utilisateur pour permettre la projection d'une interface graphique. La surface transparente est éclairée par la tranche à l'aide de nombreuses diodes infrarouge, dont la lumière reste à l'intérieur de la surface par réflexion totale (la surface se comporte comme un guide d'ondes. Au contact d'un doigt, les propriétés optiques sont modifiées (FTIR, ou *Frustrated total internal reflection*), et la caméra infrarouge située derrière la surface perçoit les endroits où a lieu le contact, permettant ainsi l'interaction digitale.

différents systèmes étudiés ne « tombent » pas précisément sur un de ces axes : une interface digitale peut également être une interface mobile, par exemple ; cependant nous décrivons chaque système dans la catégorie où il nous permet de tirer le plus de conclusions sur les besoins. Les catégories retenues sont les suivantes : interfaces tangibles, surfaces tactiles, systèmes collaboratifs, surfaces intégrées, et interfaces gestuelles. .. Dans chacun des exemples, nous notons les besoins de perception présents dans les systèmes rencontrés en supposant le cas échéant qu'ils sont réalisés à l'aide de la vision par ordinateur.

Pour chaque systèmes d'intérêt présenté nous décrirons succintement :

- les techniques d'interactions mises en oeuvre dans chaque système ;
- comment reproduire (le cas échéant) ces techniques en utilisant la vision ;
- quelles sont les contraintes que chaque technique d'interaction présente au système de perception.

À la fin de la section, nous proposons une taxonomie de ces services, afin de pouvoir structurer cet ensemble disparate et afin d'être capable de donner une définition d'un service perceptif qui soit d'une part plus formelle, et d'autre part compréhensible pour les utilisateurs concernés (Laurence, Patrick et Stanislas).

3.2.1 Interfaces tangibles

Les interfaces tangibles sont celles où l'utilisateur manipule des objets physiques qui ont un sens « immédiat » dans le monde numérique. Fitzmaurice [Fitzmaurice, 1996] en donne la définition suivante :

> « Une interface tangible fournit à ses utilisateurs l'accès à des dispositifs d'entrée multiples et spécialisée, qui peuvent servir de widgets physiques dédiés, permettant la manipulation physique et les arrangements spatiaux. »

Dans une interface WIMP, la manipulation de la souris permet de contrôler un pointeur (qui est l'acteur dans le monde numérique), pour déplacer une icône par exemple. Dans une interface tangible, l'objet manipulé peut être un avatar de l'objet numérique manipulé : l'icône numérique devient une icône physique, ou *phycon*. Il peut être un outil qui permet de donner des ordres, comme les boutons de la souris : il n'a alors pas de pendant dans le monde numérique. Enfin, il peut être une poignée anonyme qui, comme un pointeur, peut être « accrochée » à un objet numérique pour le déplacer.

Ces trois familles d'interfaces tangibles imposent des requis de perception visuelle. Nous présentons ici les deux premières sur la base de deux prototypes existants ; la dernière sera évoquée plus loin, sur l'exemple de la *Table Magique*, page 46.

3.2.1.1 Navigational Blocks

Dans *Navigational Blocks* (figure 3.4 ci-contre), un utilisateur exprime une requête en plaçant un ou plusieurs cubes dont les faces sont étiquetées (avec un nom, une date, ou un lieu) sur une surface donnée. La réponse est affichée sur un écran ; l'utilisateur peut se déplacer dans la réponse (un document contenant du texte et des images) en translatant le ou les cubes-question sur la surface active. Il existe un nombre limité de cubes (une dizaine). Au plus deux cubes peuvent être associés pour former une requête.

L'implémentation la plus directe de ce système en utilisation la vision par ordinateur serait la suivante : une caméra filme la surface active de dessus de façon à minimiser les occlusions par l'utilisateur. Chaque face des cubes est visuellement identifiable de manière unique par exemple en les munissant d'un code-barre 2D – tels ceux utilisés dans l'ARToolkit [Kato et al., 2000]). L'association entre les codes et les faces est supposée connue du système.

Pour implémenter les interactions « requête » de ce système, il est nécessaire d'identifier la face visible (pour l'utilisateur) de chaque cube, et de déterminer les positions relatives des différentes faces visibles. Ceci nous permet de formuler les trois services requis suivants :

S1. détection et identification d'un objet plan parmi des objets connus
S2. détermination des positions relatives d'objets plans

S1 et S2 correspondent à des interactions faiblement couplées, et le lieu et le lexique de l'action utilisateur et du retour d'information sont distincts. La latence minimale requise est donc élevée, de l'ordre de 250 ms. Par souci de prévisibilité, les erreurs de détection et d'identification ne sont pas tolérables : mieux vaut un échec de perception qu'une perception eronnée, qui déclencherait une requête aberrante.

Pour implémenter la navigation dans les réponses, il faut suivre les mouvements relatifs du cube sur la surface :

S3. suivi d'un objet plan en deux dimensions

Cette fois l'interaction est plus fortement couplée : une propriété physique de l'interacteur (la position du cube) est couplée au défilement dans la page d'information affichée. La latence doit donc être inférieure à 50 ms — c'est la valeur communément acceptée [Card et al., 1983]. La précision souhaitée dépend des données présentées ; pour le défilement dans une page, elle devra être centimétrique, et la stabilité statique millimétrique.

3.2.1.2 The Designer's Outpost

Dans *The Designer's Outpost* [Klemmer et al., 2001] (figure 3.5 page suivante), les utilisateurs manipulent une représentation mixte d'une structure abrorescente : une partie de la représentation est dans le monde réel, une autre est virtuelle. L'application cible est la conception de l'arbre de navigation dans un site web. Les ressources du site (pages, images) sont représentées par des photos ou des notes « post-it » placés sur un écran de grande taille. Les liens entre ressources sont représentés par des traits tirés

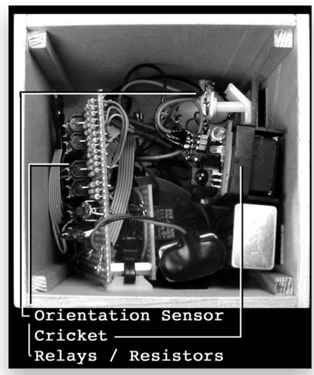


Figure 3.4 . Les cubes interactifs utilisés dans *Navigational Blocks*

Les cubes de *Navigational Blocks* [Camarata et al., 2002] sont équipés de multiples senseurs qui leur permettent de déterminer leur orientation (c'est-à-dire la face qui est sur le dessus du cube), et le contact d'une face avec une face d'un autre cube. Des électro-aimants permettent également de faire en sorte que deux cubes s'attirent ou se repoussent ; ceci n'est pas reproductible en utilisant la vision par ordinateur. Il est toutefois imaginable de fournir un retour visuel ou sonore à la place de ce retour haptique.

tâche utilisateur	technique d'interaction
poser une question simple	tourner une face d'un cube vers le haut
poser une question complexe	juxtaposer deux cubes
naviguer dans une réponse	translater le cube-question sur la surface

entre les deux ressources et affichés sur l'écran. Les tâches d'interaction élémentaires sont explicitement listées et décrites dans l'article ; elles sont résumées dans la légende de la figure.

À un instant donné, il peut exister sur la surface à la fois des ressources physique (c'est-à-dire post-its et photos représentant une ressource du site web) et virtuelles (image affichée représentant une ressource) car un arbre sauvegardé peut être restauré plus tard sans que les post-its physiques soient présents. Le travail peut alors continuer en manipulant les ressources virtuelles avec le stylo. Nous ne nous interessons ici qu'à la partie tangible de l'interaction (avec les post-its).

Les post-its doivent être détectés et reconnus par le système de vision ; mais à la différence de *Navigational Blocks*, leur identité n'est pas prédéfinie. Les utilisateurs peuvent en effet créer de nouveaux post-its, portant de nouvelles inscriptions. Par contre, ils possèdent une caractéristique commune : ils sont rectangulaires et de couleur jaune. Nous proposons les services suivants pour implémenter ce prototype :

Figure 3.5 . *The Designer's Outpost*

La partie perceptive de ce prototype [Klemmer et al., 2001] est implementé en partie en utilisant la vision par ordinateur. L'écran est un tableau *SmartBoard*, dont la surface est tactile : les utilisateurs exploitent sa surface exactement comme une tablette graphique, à l'aide de stylos spéciaux. Un seul stylo peut être utilisé à un instant donné. La detection du stylo demandant d'exercer une certaine pression, les post-its collés sur la surface du *SmartBoard* ne sont pas détectés. Une caméra a fréquence élevée et faible définition observe le tableau par l'arrière et ne perçoit que les post-its ; elle permet leur détection. Une autre caméra, à latence et résolution élevée (1500 ms, 5 millions de pixels), observe l'avant du tableau pour capturer ce qui est écrit ou dessiné sur les post-its, ainsi que les photos.
Voici les interactions élémentaires utilisées dans ce prototype :

tâche utilisateur	**technique d'interaction**
ajouter une ressource	placer un post-it ou une photo sur l'écran
supprimer une ressource	enlever un post-it de l'écran (si ressource réelle) ; passer la brosse sur la ressource (si ressource virtuelle)
déplacer une ressource	enlever et replacer un post-it (si ressource réelle) ; poser l'outil de déplacement sur la représentation de la ressource, le déplacer, puis l'enlever (si ressource virtuelle)
créer un lien	dessiner un trait au crayon entre deux ressources
effacer un lien	passer la brosse (matérielle) sur le lien
invoquer un menu contextuel	toucher un post-it
invoquer article de menu	toucher l'article de menu
annoter l'arbre	dessiner au crayon sur l'écran
sauvegarder l'arbre	toucher le bouton nommé « sauver »

S4. détection et suivi d'objets plans de couleur et de forme simple définie

La « couleur définie » pourra l'être par apprentissage (à l'initialisation du système) ; la forme pourra être explicitement définie (ici, un rectangle). Les requis de latence et de précision sont relaxés par rapport à **S1** et **S3**. Comme il n'y a pas de couplage fort, ou de translation des post-its, une latence de 500 ms est suffisante pour la détection et le suivi. Comme le positionnement des post-its est imprécis, par nature de la tâche de brainstorming, la contrainte de précision est relaxée : une précision centimétrique est suffisante. D'après les exemples de l'article cité, **S4** doit pourvoir gérer jusqu'à 20 objets dans ces conditions de performance.

S5. numérisation de l'aspect d'un objet plan

Il s'agit de fournir au client une image de l'objet, tel qu'il apparaît aux utilisateurs — en particulier à des fins de migration de l'interface de *Outpost* vers une application WIMP. La résolution géométrique doit être suffisante pour reproduire l'objet sur l'écran interactif, en restant lisible pour l'utilisateur. Une résolution de 100 dpi est satisfaisante. La contrainte de latence est relaxée, puisque l'interaction est non couplée : une latence de l'ordre de la seconde est suffisante.

S6. identification visuelle d'un objet plan

Étant donné une image d'un objet, il s'agit de retrouver son identité parmi les objets connus, ou bien de lui donner une nouvelle identité s'il n'a pas encore été observé. Ce service est bien entendu susceptible d'interagir avec **S5** pour obtenir une image de l'objet. Ce service est utilisé lorsqu'un utilisateur enlève un post-it de l'écran, puis l'y replace ultérieurement. Comme le graphe affiché doit être mis à jour en fonction de la nouvelle position du post-it, l'interaction est couplée (faiblement) ; une latence de l'ordre de 250 ms est donc requise. Comme pour **S1**, la précision (taux d'erreurs) doit être la plus faible possible, idéalement comparable celle d'un humain pour la même tâche. Un échec d'identification est préférable à une mauvaise identification : l'interface utilisateur fournit des outils pour re-coupler un post-it à une identité préexistante.

3.2.2 Interfaces digitales

La seconde famille d'interfaces rencontrée dans notre état de l'art est celle des interfaces digitales. Il s'agit de celles où les doigts sont utilisée pour manipuler des informations numériques, affichées ou projetées sur une surface : par exemple une interface graphique traditionnelle, ou des documents virtuels.

3.2.2.1 Release, Relocate, Reorient, Resize

L'organisation géographique de documents ou d'objets virtuels est l'une des applications largement explorées dans le domaine des interfaces digitales. Un pendant de cette application dans le monde physique peut être, par exemple, une tâche de classement d'une pile de documents papier sur un bureau. *Release, relocate, reorient, resize* [Ringel et al., 2004] décrit une expérience ou sont comparées différentes techniques d'interaction pour « se passer » des documents entre utilisateurs sur une interface digitale. Comme leur sujet de recherche n'est pas la technologie sous-jacente mais l'interaction, les auteurs utilisent une surface digitale performante et prête à l'emploi, la *DiamondTouch* [Shen et al., 2004].

Dans cette expérience, que nous nommons *R4*, quatre techniques d'interaction sont comparées. Pour nous elles sont équivalentes (c'est-à-dire en termes de besoin de services perceptifs) : un document virtuel projeté sur la surface peut subir une translation, une rotation, ou une mise à l'échelle, selon le geste effectué par l'utilisateur. Deux services sont nécessaires pour reproduire l'expérience en vision par ordinateur :

S7. suivi d'un doigt par utilisateur

Il existe de nombreuses approches pour implémenter ce service — une description des ces approches est présente dans [Letessier, 2003] et [Letessier et Bérard, 2004].

L'objectif d'une interface digitale comme *R4* est de provoquer la sensation d'immersion chez l'utilisateur, c'est-à-dire d'effacer l'impression de manipuler des documents virtuels. Cette immersion ne peut être parfaite en utilisant la vision « pure » — il manque le retour tactile ; nous savons cependant qu'une immersion optimale peut être atteinte en maximisant la performance du système de vision. L'interaction étant ici fortement couplée, la latence du suivi doit être inférieure à 50 ms ; sa précision et sa stabilité statique, millimétrique. Un doigt par utilisateur (jusqu'à 4 utilisateurs) doit être suivi. Les utilisateurs ne font qu'un geste, celui du doigt pointé. Enfin, le système est utilisé dans un environnement contrôlé, ou l'éclairage est compatible avec l'utilisation confortable d'un videoprojecteur.

S8. identification approximative du propriétaire d'un doigt.

La tâche d'identification « absolue » (déterminer le nom de la personne en train d'interagir) ne peut pas être implémentée de manière réaliste en vision par ordinateur, c'est-à-dire dans des conditions peu contrôlées et en temps réel. Par contre, il est possible de fournir une information comparable à celle fournie par la table *DiamondTouch* utilisée pour l'article *R4* : par exemple en fournissant à l'application cliente la position du corps de l'utilisateur par rapport à la surface d'interaction ; c'est-à-dire, une identification arbitraire, mais cohérente dans le temps, des utilisateurs associés à chaque doigt.

3.2.2.2 Visual Touchpad

L'objectif de ce projet [Malik et Laszlo, 2004] figure 3.6 est de construire une surface digitale plus riche que les surfaces habituelles (par exemple : tablet PC, *SmartSkin* [Rekimoto, 2002], ou *DiamondTouch*) pour fournir de nouvelles techniques d'interaction. En particulier, les auteurs souhaitent pouvoir exploiter l'interaction bimanuelle ou a plusieurs doigts par main, l'orientation des doigts, et l'information de survol (quand les doigts sont à proximité de la surface sans la toucher) pour réaliser des techniques d'interaction innovantes.

Le service de vision par ordinateur requis est fonctionnellement très différent de **S7**.

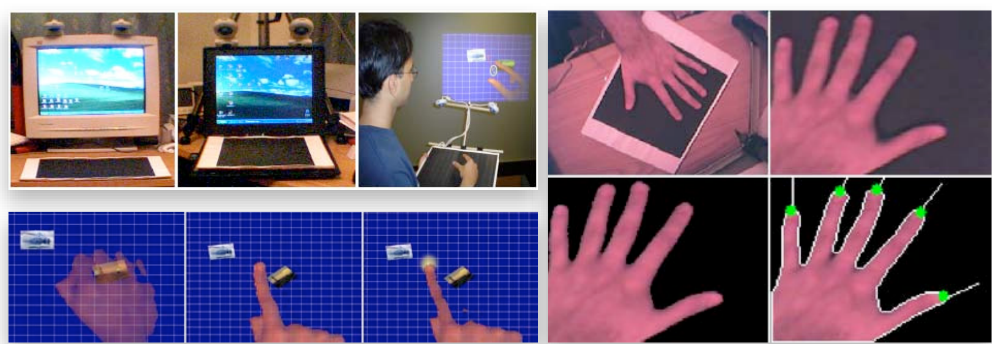


Figure 3.6 . La tablette tactile virtuelle *Visual Touchpad*

Ce prototype [Malik et Laszlo, 2004] reproduit en vision par ordinateur une tablette graphique ordinaire, et fournit des fonctionnalités supplémentaires : multiples points de contact, orientation des doigts, et information de distance lorsqu'il y a survol sans contact.
Un morceau de carton noir, non équipé, est filmé par une paire de caméras (en haut à gauche). Un algorithme de détection de doigts simple [Wu et al., 2000] mais peu robuste [Letessier, 2003] est utilisé dans chaque vue : rectification, seuillage de luminosité, extraction de contours, détection des maxima de courbure négative (à droite). Lorsque la disparité entre deux détections correspondantes dans chaque vue est sous un seuil fixé, le système considère qu'il y a contact avec le *touchpad*, ce qui permet d'utiliser la surface anodine comme surface tactile (en bas à droite).

Il n'y a qu'un utilisateur (au plus deux mains) ; tous les doigts de l'utilisateur doivent être suivis ; l'orientation des doigts et leur distance à la surface doivent être mesurées (de manière continue ou discrète) ; enfin, la scène vue par la caméra est contrainte, les doigts se déplacent sur un fond noir.

Nous notons ce services :

> **S9.** suivi avec orientation des doigts d'un utilisateur

Par contre, il est très similaire en termes non fonctionnels : la latence requise est également de 50 ms. La précision doit être millimétrique, sauf selon l'axe vertical (l'axe de profondeur, perpendiculaire à la surface du *touchpad*), ou la précision est discrète (avec trois valeurs : contact, proche, ou éloigné). Les autres considérations restent également valides.

3.2.2.3 RoomPlanner

Le prototpe *RommPlanner* [Wu et Balakrishnan, 2003] (figure 3.7) est une application d'agencement de mobilier de bureau, conçue pour être utilisée par plusieurs utilisateurs collaborant sur une interface digitale. De nombreuses techniques d'interaction sont mises en place pour manipuler des objets ou des groupes d'objets, invoquer des menus contextuels, ou afficher des méta-informations.

Ces techniques de manipulation peuvent être scindées en deux catégories. La première est constituée des interactions digitales *per se*, ou un utilisateur désigne des objets précis avec un ou plusieurs doigts (éléments de mobilier, menus ou sous-menus). En langage WIMP, il clique, double-clique, ou effectue un glisser-déposer avec un ou plusieurs doigts. Ici encore le besoin est différent de celui auquel répond S7, et S7 n'est pas fonctionnellement satisfaisant. S9 non plus, puisque plusieurs utilisateurs doivent pouvoir interagir sur la surface. En outre l'orientation et l'élévation des doigts ne sont pas pertinentes ici. Nous introduisons donc le besoin :

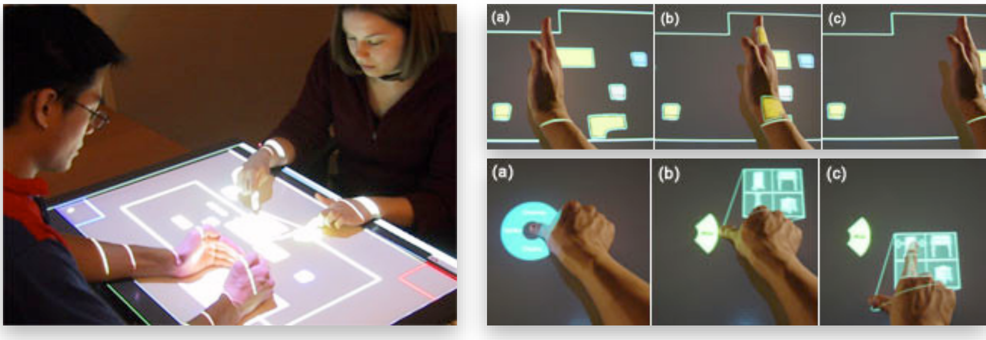


Figure 3.7 . L'application *RoomPlanner* sur la table *DiamondTouch*.

La table interactive *DiamondTouch* [Dietz et Leigh, 2001] est constituée de deux grilles d'antennes perpendiculaires. Lorsqu'un utilisateur est couplé à la table (en étant assis sur un recepteur dédié), les variation de capacité dans le circuit ainsi formé permettent de détecter la proximité et le contact de l'utilisateur avec la table, et la projection de la zone de contact sur l'axe horizontal et sur l'axe vertical de la table. En multiplexant les détection dans le temps, le système est capable de distinguer plusieurs utilisateurs (jusqu'à quatre dans le prototype).
Ci-dessous, un extrait des techniques d'interaction utilisées.

tâche utilisateur	**technique d'interaction**
déplacer un objet	toucher l'objet, glisser le doigt jusqu'à la destination, relever le doigt
invoquer le menu contextuel	toucher l'objet d'intérêt deux fois, rapidement (*double-tap*)
regrouper un ensemble d'objets	placer les deux mains perpendiculairement à la surface, en la touchant, puis les rapprocher

S10. suivi des doigts de plusieurs utilisateurs.

Les requis non fonctionnels sont également légèrement différents : au vu des techniques d'interaction proposées, seuls le pouce et l'index de chaque main doivent être suivis. D'après les auteurs, la collaboration peut avoir lieu avec plus d'utilisateurs que dans les exemples de l'article (jusqu'à 5).

La seconde catégorie d'interactions est constituée des interactions gestuelles effectuées avec les mains, au contact de la surface. Six gestes sont définis, soit statiques (une posture d'une ou deux mains déclenche une action), soit dynamiques (une séquence de postures déclenche une action). En laissant de côté le traitement de l'aspect dynamique (il pourrait par exemple être délégué à l'application), le service correspondant est noté :

S11. suivi et classification de la posture de chaque main visible

Il est destiné à des interactions couplées. Par contre, le requis de précision est plus relaxé : il est de l'ordre de grandeur de la précision que l'utilisateur peut atteindre, soit 1 cm. Comme noté plus haut, jusqu'à 10 mains peuvent être visibles.

3.2.3 Systèmes collaboratifs

Les chercheurs du domaine du travail collaboratif assisté par ordinateur (*Computer supportated collaborative work*, ou CSCW) ont produit de nombreux systèmes expérimentaux. Le travail collaboratif est un besoin régulièrement exprimé par le monde industriel. Dans le cas des activités quotidiennes d'une entreprise ou d'une équipe de recherche par exemple, la collaboration est synchrone et coprésente : plusieurs personnent travaillent ensemble autour du même support ou du même outil. Au sein du domaine CSCW, le collecticiel sur affichage partagé est une axe de recherche très actif (*single display groupware*) : il s'agit de construire des systèmes dotés d'une interface graphique située en un lieu unique et capable de répondre aux stimuli simultanés de plusieurs utilisateurs. C'est cette piste, explorée par des systèmes technologiquement innovants, qui nous interesse ici.

Nous présentons deux systèmes dans cette section : *Caretta* et la *Table Magique*. Nous aurions pu y inclure *The Designer's Outpost* car c'est également un système collaboratif. Réciproquement, *Caretta* est également une interface tangible, et aurait pu être traité dans la première section. Nous avons préféré les décrire ainsi, car leurs besoins propres sont ainsi plus apparents.

Les besoins fonctionnels en termes de services rencontrés peuvent donc être similaires à des besoins évoqués précedemment. Cependant certaines caractéristiques seront différentes, celles liées à la collaboration. En particulier, le nombre d'agents interactifs qui peuvent évoluer simultanément est plus élevé, ce qui impose des contraintes de performance plus lourdes sur le système de perception.

3.2.3.1 Caretta

Caretta [Sugimoto et al., 2004] (figure 3.8 ci-contre) est un système collaboratif de simulation d'urbanisme à l'image du jeu *Maxis SimCity*. Il intègre un espace personnel (l'écran d'un PDA) et un espace partagé (la table interactive). Les utilisateurs manipulent les mêmes informations sur les deux espaces, tout en ayant la possibilité d'expérimenter sur une version privée de la simulation. Nous nous focalisons sur l'interaction sur la table — pas à celle avec les PDA, ou entre les PDA et la table.

D'après cette description, et celle des techniques d'interaction mises en jeu, les besoins peuvent être similaires à ceux du *Designer's Outpost*, en particulier ceux satisfaits par les services **S4** et **S6**. En fait il existe deux différentes importantes :

- sur le plan fonctionnel, ce n'est pas l'identité propres des objets d'interaction (les tuiles) qui est pertinente, mais leur appartenance à une classe (telle tuile répresente une habitation, telle autre une zone commerciale). En outre, seule la détection et la localisation des tuiles sont nécessaire ; il est inutile de les suivre.

– sur le plan non fonctionnel, là où *Outpost* requiert un service capable de gérer une dizaine d'objets, *Caretta* doit pouvoir fonctionner avec un nombre de tuiles de l'ordre de la centaine.

Le service requis est donc différent :

> **S12.** détection, localisation, et classification d'objets parmis des classes connues.

En vision par ordinateur, l'appartenance à une classe peut se faire parmi des classes connues par leur aspect, par similarité, ou par toute autre méthode d'identification. Il est possible d'utiliser, comme proposé plus haut pour reproduire *Navigational Blocks*, un système de *tags* identifiant chaque classe, imprimés sur la face supérieure de chaque tuile.

Comme l'interaction avec les tuiles est faiblement couplée, le requis de latence est de l'ordre de 250 ms. La précision et la stabilité statique doivent être de la taille des tuiles et de la grille de « jeu », soit 1 cm.

3.2.3.2 La Table Magique

La *Table Magique* [Bérard, 2003] figure 3.9 page suivante est un tableau blanc augmenté ayant pour objectif le support de la communication entre humains lors de réunions créatives. Elle sert de portail entre le monde physique et le monde numérique : les utilisateurs peuvent dessiner ou écrire à l'aide de stylos ordinaire, numériser l'information, puis manipuler les objets virtuels correspondants.

L'interaction se fait par le biais de nombreux jetons colorés, que les utilisateurs déplacent sur la surface de la table, et qui permettent de créer et manipuler des *patches*

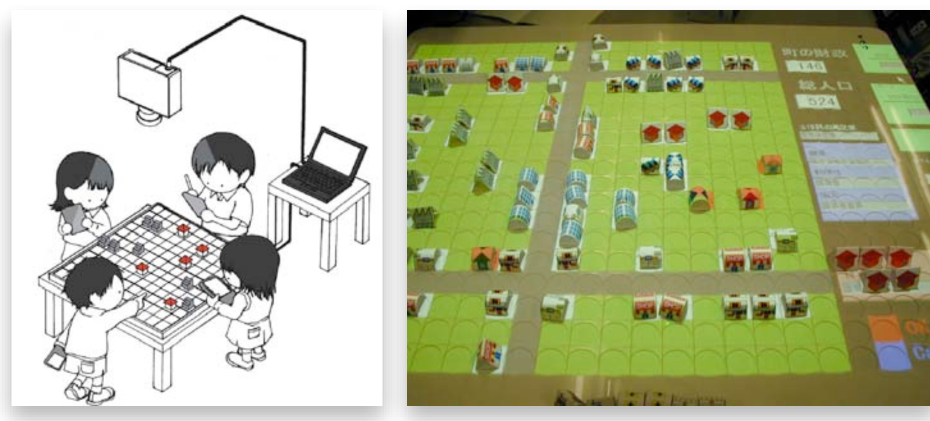


Figure 3.8 . Le jeu collaboratif multimodal *Caretta*

Dans ce système [Sugimoto et al., 2004], toutes les tuiles sont munies d'une puce passive RFID (*Radio Frequency Identification*). Chaque puce porte un identifiant unique ; une base d'identifiants permet en outre de déterminer à quelle classe chaque tuile appartient (type de bâtiment, tuile spéciale). Le plateau, nommé E^2*board*, est en fait une matrice de 20x24 récepteurs radio ; chaque récepteur est connecté à l'une des 4 unités de calcul du plateau, elles-mêmes connectées à un controlleur, puis à l'ordinateur qui pilote le plateau. Ainsi, l'application cliente peut déterminer, à tout instant, quelle tuile est sur chacune des cases du tableau.
D'après les auteurs, les données du plateau sont rafraîchies à 10 Hz, et obtenues avec une latence de l'ordre de 50 ms.

tâche utilisateur	**technique d'interaction**
ajouter un bâtiment	placer la tuile sur une case du plateau
supprimer un bâtiment	retirer la tuile du plateau
invoquer article de menu	placer une tuile « action » sur la case de l'article de menu

(morceaux) d'encre virtuelle numérisée. Deux services de perception sont nécessaires. Le premier concerne la perception des jetons : ici **S4** page 43 convient presque parfaitement. Seul le requis de latence est plus élevé, car cette fois l'interaction est fortement couplée. En effet la position d'un jeton peut être couplée à celle d'un *patch* en cours de translation, ou à un coins d'un zone de numérisation en cours de définition. Nous notons ce service plus contraint **S4'**.

Le second service requis concerne la création des *patches*, c'est-à-dire la numérisation de l'encre physique présente sur une zone donnée de la table. Cette fois c'est **S5** qui est suffisant pour remplir ce rôle. La seule différence est que les dimensions de la zone à numériser sont potentiellement plus grandes. Là ou dans *Outpost* les zones à numériser étaient toujours de 50×50 mm, ici elles peuvent être aussi grandes que la table elle-même ; soit 750×500 mm environ -- le requis de précision restant le même. Nous notons ce service de capture à haute défintion **S5'**.

Figure 3.9 . La *Table Magique* , un tableau blanc interactif pour l'augmentation des réunions informelles.

Ce dispositif est un tableau blanc ordinaire sur lequel les utilisateurs écrivent et dessinent avec des stylos effaçables. Des jetons de plastique rouge permettent de copier l'information du monde physique au monde numérique : en mettant deux jetons en contact, puis en les écartant, l'utilisateur définit un rectangle de sélection. Le texte et les dessins à l'interieur sont scannés par une caméra, puis projetés à l'identique ; la projection est nommée un « patch ». Ils peuvent alors être éffacés physiquement : la copie numérique projetée persiste. Les jetons servent également d'ancres qui peuvent être accrochées aux informations numériques projetées, pour les déplacer ou les transformer (en échelle et orientation).
La perception des jetons est implémentée en vision par ordinateur par un logiciel de detection et de suivi d'objets suivant un modèle de couleur. Le système est capable de suivre de nombreux jetons permettant ainsi de créer une application multi-utilisateurs. La latence du suivi est de l'ordre de 70 ms.

tâche utilisateur	**technique d'interaction**
créer un patch	amener des jetons en contact, les écarter pour englober la zone à scanner, puis les laisser immobiles pendant 2 secondes
lier un jeton à un patch	placer le jeton sur le patch, attendre 200 ms le retour visuel
détacher un jeton d'un path	masquer le jeton pendant au moins 500 ms
transformer un patch	lier un ou deux jetons au patch ; les déplacer ; les détacher du patch
supprimer un patch	transformer le patch pour amener sa taille à zéro

3.2.4 Nouvelles surfaces d'interaction

L'informatique évanescente (*disappearing compunting*) fait disparaître l'ordinateur (le triplet clavier, écran, souris évoqué au chapitre précédent) en l'intégrant dans l'environnement de manière plus ou moins transparente. L'informatique ubiquitaire (*ubiquitous computing*) rend les capacités de calcul et d'information accessibles depuis des lieux multiples, éventuellement de manière mobile.

Ces deux axes de recherche font apparaître de nouvelles surfaces d'interaction possédant de nouvelles propriétés. Certaines interfaces digitales ou tangibles en font partie : la *Table Magique* pourrait par exemple être étiquetée comme système évanescent, au titre qu'elle fournit de l'interaction avec le monde numérique d'une manière transparente et intégrée dans un environnement classique.

D'autres systèmes interactifs font intervenir des surfaces plus originales, aux propriétés « extrêmes » en comparaison avec l'informatique classique : surfaces de dimensions très grandes ou très petites, fragmentées dans l'environnement, ou mobiles. Sans prétendre décrire exhaustivement l'utilisation de surfaces interactives dans ces deux domaines de recherche, nous présentons ici trois systèmes et trois pistes qui apportent des besoins de perception nouveaux.

3.2.4.1 Surfaces interactives intégrées

Everywhere Displays [Sukaviriya et al., 2003] (figure 3.10 page suivante) a pour objectif l'augmentation d'un invironnement donné par projection d'informations contextuelles et de surfaces interactives en de multiples lieux d'un environnement donné. Le prototype décrit dans l'article cité est installé dans un *mock-up* de grand magasin. Le système est capable de suivre les utilisateurs, et de leur fournir l'information et l'interaction pertinentes selon le rayon du magasin où ils se trouvent, en dirigeant l'interface vers la surface connue la plus proche d'eux. Dans l'exemple une seule interface dirigeable est disponible, et un seul utilisateur peut interagir avec cet environnement.

Pour diriger l'interface, il est nécessaire de savoir si un utilisateur est proche d'un lieu d'intérêt. La localisation absolue n'est pas nécessaire ici, une information de distance entre ce lieu et l'utilisateur est suffisante. Le lieu d'intérêt peut être colocalisé avec le capteur. D'où le service :

S13. détection et localisation relative de personnes

Pour le projet décrit, une latence de l'ordre de la seconde est sufisante. L'interface doit en effet avoir été dirigée vers la surface adaptée avant que l'utilisateur y ait porté son attention. La précision requise est également grossière, de l'ordre du mètre. La robustesse est contrainte de manière asymétrique : peu importe qu'une interface s'affiche alors qu'aucun utilisateur ne la regarde, du moment qu'aucune interface ne « disparaît » de la vue d'un autre. D'après l'étude utilisateur des auteurs, c'est l'un des défauts importants du prototype tel qu'il a été déployé.

Pour implémenter **S13** en vision par ordinateur, Laurence peut équiper chaque lieu actif d'une caméra (même de résolution et fréquence faible), dont le champ de vision est susceptible de contenir l'utilisateur. Un détecteur de visage peut alors être utilisé pour détecter la présence d'un utilisateur et estimer sa distance. **S13** pourrait également être le support de l'interaction implicite dans IPAD (figure 2.7 page 30).

Pour l'interacion tangible avec les panneaux d'information comportant des sliders physiques, le service utilisé pour les jetons de la *Table Magique* (**S4'**) est adapté. Pour le dernière forme d'interaction, l'activation de boutons virtuels par occlusion avec la main, il pourrait être possible d'utiliser l'un des services vus pour les interfaces digitales (**S7**, **S9**, **S10**, ou **S11**). Cependant ces services fournissent une information trop riche : nul besoin, par exemple, de connaître l'orientation des doigts de l'utilisateur. En outre, la contrainte de robustesse est ici plus importante, car l'environnement visuel est plus complexe. Nous proposons donc un nouveau service :

S14. détection de l'occlusion d'un widget projeté par un doigt

Figure 3.10 . Le projet d'interaction sur surfaces dirigeables *Everywhere Displays*

Dans ce projet [Sukaviriya et al., 2003], un triplet caméra, projecteur, miroir orientable permet de projeter une interface graphique sur de multiples surfaces, dans le contexte applicatif d'un magasin (a). Un système de vision permet de rendre interactifs des widgets physiques : ici (b,c) des *sliders* (barres de défilement) tangibles, en bois, permettent de défiler dans l'information projetée. Ailleurs, un affichage n'est qu'informatif (d). Enfin, sur une table, il devient interactif (e) en rendant certaines zones sensibles aux occlusions par une main — comme des boutons WIMP virtuels.

tâche utilisateur	**technique d'interaction**
obtenir de l'information sur	s'approcher du rayon / de l'article
un article	
défiler dans l'information	déplacer le *slider* physique
projetée sur un panneau	
activer un bouton projeté	placer la main sur le bouton

La latence de ce service doit être de l'ordre de 250 ms. Il doit être robuste aux occlusions partielles involontaires, comme aux occlusions totales involontaires (par exemple lorsque le « client » se penche devant l'interface). Remarquons que d'autres projets exposent également ce besoin (par exemple [Fails et Olsen Jr., 2002]).

3.2.4.2 Grandes surfaces interactives

i-LAND [Streitz et al., 1999] est un projet d'environnement de travail augmenté (figure 3.11). Le prototype comporte plusieurs objets augmentés grâce à l'intégration de capacités de perception, de calcul, et d'affichage dans des objets anodins (*roomware*) liés entre eux par un réseau sans fil. Il s'agit de chaises équipés de tablet PC (*CommChairs*), d'une table interactive électronique (*InteracTable*), et d'un mur interactif électronique (*Dynawall*). Notre intérêt se porte ici sur la table et le mur.

InteracTable est très similaire en fonctionnalités et en dimensions à la table Mitsubishi *DiamondTouch* utilisée pour déployer *RoomPlanner* (figure 3.7 page 45). Par contre, si *Dynawall* possède un besoin de perception fonctionnellement similaire, il possède une différence implortante : sa surface est environ douze fois supérieure. Dans les deux cas, le besoin en termes de service de vision est **S10**. Il est probable que le même logiciel ne pourra pas satisfaire le besoin dans les deux cas. Patrick, l'intégrateur de service, peut cependant multiplier les caméras et les instances du service **S10** pour implémenter un nouveau logiciel de perception sur surface interactive de grandes dimensions.

3.2.4.3 Surfaces interactives portables

Dans le cadre du *Peephole Displays* [Yee, 2003] (figure 3.12 page suivante), un ordinateur de poche est augmenté d'un capteur permettant d'utiliser sa position dans l'espace comme entrée dans un logiciel : déplacer l'ordinateur dans le plan de l'écran du PDA permet à l'utilisateur de contrôler la position de la fenêtre physique (le PDA) sur le document virtuel qui est plus grand que le PDA. Déplacer le PDA dans l'axe perpendiculaire au plan permet de changer de mode. Yee expérimente avec différent capteurs (PDA relié à une souris, localisateur magnétique) mais aucun ne permet un usage réellement mobile du système et auncun n'est basé sur la vision. Il paraît naturel d'envisager une solution basée sur la vision grâce à une caméra généralement présente sur les derniers modèles de PDA.

De manière similaire, le *PDS* [Borkowski et al., 2005] (figure 3.13 page suivante) est

Figure 3.11 . *i-LAND*, une salle de réunion augmentée offrant de nombreux dispositis d'interaction de nouvelle génération.

InteractTable (à gauche) et *Dynawall* (à droite), sont probablement conçus en juxtaposant plusieurs écrans rétroprojetés tactiles de *Smart Technologies*. Ils permettent l'interaction digitale ou au stylet, avec un ou deux utilisateurs simultanés (par sous-écran, dans le cas de *Dynawall*). L'article ne fournit aucune évaluation des performances utilisateur des différents dispositifs.

une surface interactive portable, implémentée en vision par ordinateur, dont les déplacements sont interprétés par un système qui s'assure que l'interface reste projetée sur la surfaces. L'auteur envisage également d'utiliser ces déplacements pour implémenter un *peephole display* ou, en tout cas, afficher des interfaces différentes suivant la position et l'orientation de la surface.

Ces deux systèmes partagent superficiellement le même requis perceptif : un suivi d'objet en trois dimensions par rapport au capteur. Dans le premier cas, c'est la position de l'utilisateur ou d'un objet fixe de l'environnement par rapport à la caméra (PDA) qui est pertinente, et l'orientation du dispositif n'a pas d'importance. Dans le second cas le suivi est relatif au projecteur (en fait, au couple caméra-projecteur) et l'orientation est primordiale puisqu'elle permet de projeter une image rectifée sur le *PDS*. D'où les deux services :

> **S15.** suivi en trois dimensions d'un objet relativement au capteur
>
> **S15'.** suivi en trois dimensions de la position et de l'orientation d'un objet bien connu, relativement au capteur

Les requis non fonctionnels, par contre, sont similaires : l'interaction étant fortement couplée, la latence du système perceptif doit être inférieure à 50 ms.

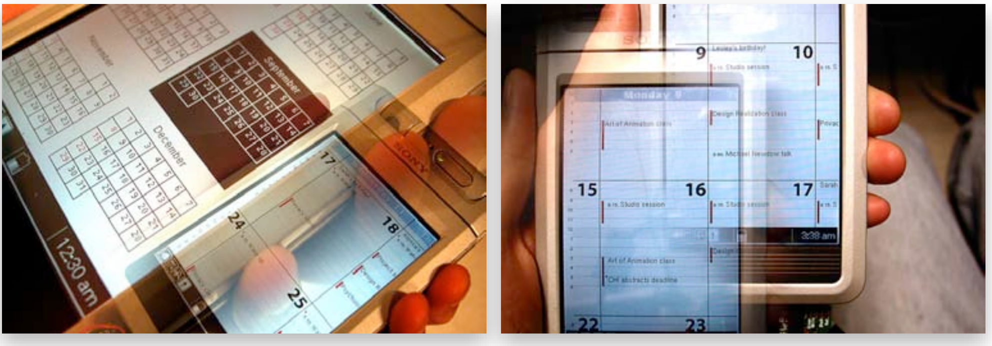


Figure 3.12 . *Peephole Displays*

Un accéléromètre 3D est installé sur un ordinateur de poche. Les données qu'il fournit sont intégrées deux fois pour fournir une information de position relative de l'appareil. Ceci est utilisé pour la navigation discrète entre espaces de travail (suivant l'axe vertical, à gauche), ou la navigation « peephole », continue, dans le même espace (dans le plan horizontal, à droite).

Figure 3.13 . Le *PDS* (Portable Display Surface), une surface anodine augmentée.

D'après [Borkowski et al., 2005]. Un couple caméra-projecteur orientable (à gauche) permet de transformer une surface anodine en carton en surface interactive portable (à droite, en bas), dans tout l'hémisphère autour de l'appareil. Ce système de vision par ordinateur est décrit dans les chapitres suivants. Des applications interactives, comme l'application ContACT de réalisation de présentations, peuvent ensuite être utilisées sur cette surface (à droite, en haut).
D'autres approches utilisant la vision existent : par exemple [Lee et al., 2005] équipe la surface de senseurs.

En vision par ordinateur, l'implémentation de ces deux services sera très différente. Dans le premier cas la caméra est montée sur la surface interactive elle-même et observe l'utilisateur ; un algorithme de suivi par corrélation (comme dans une souris optique) semble adapté. Un exemple d'implémentation en vision est décrit dans [Wang et Canny, 2006]. Dans le second cas, la caméra est colocalisée avec le projecteur : c'est donc la surface qu'il s'agira de déctecter et suivre. Un exemple d'implémentation utilisant en partie la vision est présenté par [Lee et al., 2005].

3.2.5 Interaction gestuelle

Pour accroître la sensation d'immersion, ou permettre d'exploiter les comportements naturels comme modalité d'interaction avec une application, de nombreux projets construisent des interfaces « gestuelles ». Il s'agit des interfaces où des mouvements du corps, ou d'une partie du corps, vont être exploités. En général ces interfaces sont du type « miroir video » : une image de l'utilisateur est affichée sur l'écran devant lui et l'utilisateur s'identifie à cette image pour interagir.

Plusieurs auteurs ont exploré l'utilisation des mouvements de la tête pour le contrôle d'une application (figure 3.14). Dans *Head Gestures* [Morency et Darrell, 2006], l'objectif est d'ajouter une modalité au système : pour valider une boîte de dialogue WIMP, l'utilisateur peut soit cliquer sur « Ok » à la souris, soit hocher la tête. Dans la *Fenêtre Perceptuelle* [Bérard, 1999a] il est possible d'utiliser la têtes en remplacement d'une molette de souris 2D. Dans un grand document tant que l'utilisateur maintient une touche du clavier enfoncée les mouvements (en deux dimensions) de sa tête sont couplés au défilement dans le document. Dans les deux cas, le requis de perception est :

S16. suivi 2D des mouvements de la tête d'un utilisateur

Pour la *Perceptual Window*, les considérations d'usage imposent une latence maximale de 50 ms. La précision et la stabilité statique requise n'on pas été évalués par les différents auteurs, mais nous estimons qu'une précision de l'ordre de 5 mm est nécessaire (de l'ordre de la stabilité de la tête humaine). Pour *Head Gestures*, ces

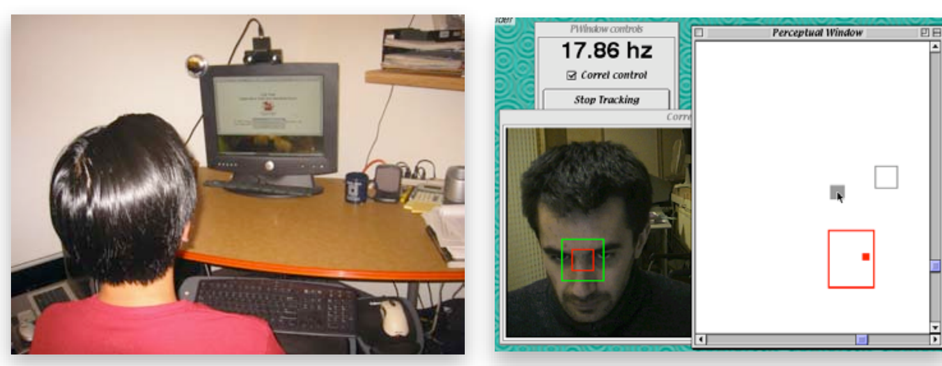


Figure 3.14 . Deux projets utilisant les mouvements de la tête comme entrée

Head Gestures (à gauche) utilise une paire de caméras stereo et des algorithmes complexes fondés sur un modèle géométrique de la tête et sur l'apparence locale pour obtenir un suivi complet de la tête de l'utilisateur (i.e. suivi à 6 degrés de liberté). Ce résultat sert à déterminer la vitesse angulaire de la tête suivant chaque axes ; cette vitesse est injecté dans un classificateurs (stochastique, avec apprentissage, utilisant des SVM), pour déterminer si l'utilisateur fait un geste d'acquiescement ou de négation avec la tête.
La Fenêtre Perceptuelle (à droite) utilise une seule caméra, et effectue un suivi 2D par ZNCC (corrélation croisée normalisée) de la zone située entre les yeux, qui est suffisemment visuellement invariante par rotation. En contrepartie de l'utilisation de cet algorithme simple et rapide, l'initialisation doit être faite manuellement par l'utilisateur. Dans l'exemple, la position relative 2D obtenue est utilisée pour défiler dans un document.

contraintes sont relaxées : une latence plus élevée (500 ms) est tolérable. Ce service moins contraint est noté **S16'**.

L'implémentation de ce service en vision est possible — elle a été réalisée dans les deux systèmes présentés. Cependant une implémentation plus autonome est requise.

D'autres utilisent les mouvements du corps entier pour l'interaction faiblement ou fortement couplée. Pour déterminer l'activité de l'utilisateur comme information contextuelle, [Jaimes, 2006] détermine a posture à partir de sa silhouette visible. Pour réaliser plusieurs jeux interactifs dans une salon ludique sur les nouvelles technologies, [Krueger et al., 1985] reprojette la silhouette comme un ombre, et y affiche des personnages animés qui interagissent avec les contours de la silhouette (c'est le cas de *Critter*), ou l'utilise comme pointeur.

Dans les deux cas le besoin est fonctionnellement le même. Par contre les contraintes sont différentes : dans le premier cas une latence élevée, de 1000 ms, est acceptable ; dans le second, l'interaction est fortement couplée et la latence doit être faible. La précision dépend aussi de la tâche : pour le premier cas, les auteurs utilisent une résolution basse (QVGA, 320×240 pixels) pour la silhouette ; dans le second, une résolution moyenne (PAL, 720×576 pixels) est utilisée (car la silhouette est reprojetée à l'échelle 1).

Les deux services sont notés :

S17. détermination de la silhouette visible d'un utilisateur (latence élevée, basse définition)

S17'. détermination de la silhouette visible d'un utilisateur (latence faible, haute définition)

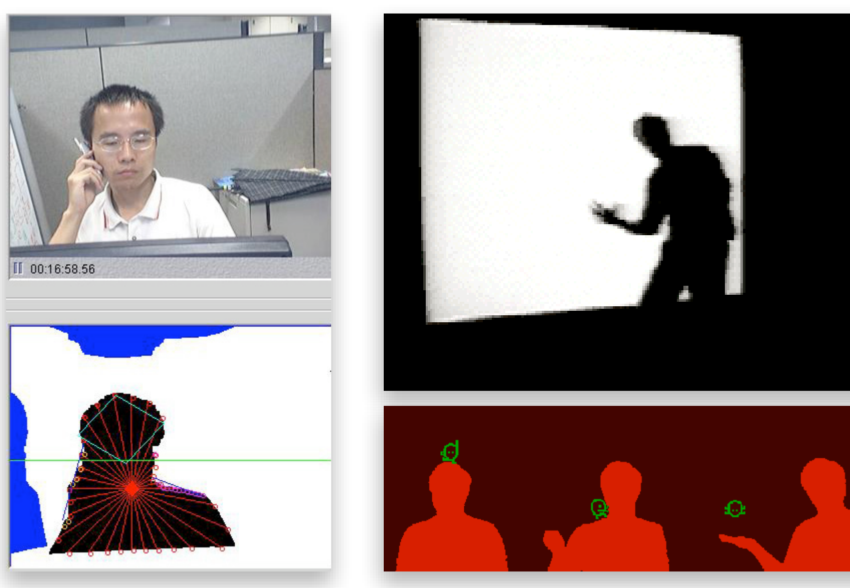


Figure 3.15 . Deux projets utilisant la silhouette du corps comme entrée.

À gauche, un système de vision détermine la silhouette, puis la modélise comme un vecteur des distances entre le barycentre de la silhouette et son contour, pour un ensemble discret d'angles ; ces données sont fournies à un classificateur probabiliste qui en déduit une prédiction de l'activité courante de l'utilisateur.
À droite, dans le cadre du système *VideoPlace* de Krueger, le jeu *Critter*. Le contour de la silhouette est déterminé par simple seuillage (l'utilisateur est très éclairé, dans un environnemnt sombre) ; la silhouette est reprojetée devant lui, comme un miroir.

3.3 Synthèse : une taxonomie des besoins et services

Au cours de ce chapitre nous avons décrit et étudié différents systèmes existants, déjà implémentés, dont la perception est ou pourrait être implémentée en vision par ordinateur. Nous avons extrait de la description fournie par leurs auteurs un ensemble de requis fonctionnels en termes de services de vision par ordinateur permettant de reproduire ces systèmes. Pour chaque service particulier nous avons isolé des requis non fonctionnels (en termes de latence, de fréquence, de précision, etc).

Malgré des similarités entre les besoins, nous avons défini 23 services distincts soit par des fonctionnalités différentes soit des requis non fonctionnels différents (figure 3.16).

Il est donc clair que satisfaire précisément chacun des besoins dans une boîte à outils de perception conduirait à une explosion du nombre de services fournis. Remarquons toutefois que plusieurs systèmes ont des besoins « inclus » l'un dans l'autre, c'est-à-dire que les deux sont fonctionnellement proches, mais l'un est moins contraignant que l'autre. Par exemple **S4** (suivi des post-it de *Outpost*, page 43) est inclus dans **S4'** (suivi des jetons de la *Table Magique*, page 48). Ceci indique qu'il peut être possible de factoriser certains services fonctionnellement proches.

Il apparaît donc nécessaire d'obtenir un moyen plus formel de décrire un besoin de perception et de minimiser le nombre de services requis tout en maximisant le nombre de systèmes qu'il est possible de réaliser à partir de ces services. Nous proposons pour ce faire de définir une taxonomie des services de perception visuelle pour l'interaction.

3.3.1 Factorisation des services

3.3.1.1 Qualité de service

La similarité la plus marquante entre tous les services concerne les contraintes non fonctionnelles qui s'y appliquent. Nous prenons la convention de nommer *qualité*

S1	détection et identification d'un objet plan parmi des objets connus
S2	détermination des positions relatives d'objets plans
S3	suivi d'un objet plan en deux dimensions
S4	détection et suivi d'objets plans de couleur et de forme simple définie
S4'	ibid, avec latence faible
S5	numérisation de l'aspect d'un objet plan (basse définition)
S5'	numérisation de l'aspect d'un objet plan (haute définition)
S6	identification visuelle d'un objet plan
S7	suivi d'un doigt par utilisateur
S8	identification du propriétaire d'un doigt
S9	suivi avec orientation des doigts d'un utilisateur
S10	suivi des doigts de plusieurs utilisateurs.
S11	suivi et classification de la posture de chaque main visible
S12	détection, localisation, et classification d'objets plans parmi des classes connues
S13	détection et localisation relative de personnes
S14	détection de l'occlusion d'un widget projeté par un doigt
S15	suivi en trois dimensions d'un objet connu relativement au capteur
S15'	suivi en 3D de la position et de l'orientation d'un objet bien connu, relativement au capteur
S16	suivi 2D des mouvements de la tête d'un utilisateur
S16	suivi 2D des mouvements de la tête d'un utilisateur
S17	détermination de la silouhette visible d'un utilisateur
S17'	détermination de la silouhette visible d'un utilisateur (latence faible, haute définition)

Figure 3.16 . Liste des services rencontrés lors de l'état de l'art des systèmes interactifs de nouvelle génération.

de service les performances d'un système perceptif suivant chacun des critères non fonctionnels : nombre d'agents, latence, précision, stabilité statique, autonomie, et robustesse.

Le *nombre d'agents* est une fourchette dans laquelle le service doit fonctionner en respectant les autres critères de qualité. Certaines valeurs sont particulières : si la borne inférieure est 0, le servicce fonctionne correctement si aucun agent n'est présent ; si la fourchette est de la forme *k*-*k*, ceci signifie que le service peut supposer qu'exactement *k* agents sont présents dans la vue.

Le requis de latence dépend du couplage entre l'action de l'utilisateur et le retour du système interactif. Lorsque le couplage est fort la latence doit être inférieure en moyenne à 50 ms [Card et al., 1983] avec un écart-type maximum de 80 ms [Watson et al., 1998]. C'est le cas lorsque un mouvement de l'utilisateur est lié directement au mouvement d'une information projetée, colocalisée ou non. Lorsque le couplage est faible, tel que le retour d'information due au click sur un bouton (**S14**), nous estimons la latence requise à environ 250 ms. Lorsqu'il n'y a pas de couplage explicite, comme pour la numérisation des post-it de **S5**, la latence maximale requise dépend du delai avant la future exploitation de la perception par le système. Dans le cas de designer's outpost elle est de l'ordre de 1000 ms.

La précision à atteindre dépend de la tâche et du type d'interaction. Pour les tâches de localisation, la précision dépend des dimensions de l'objet localisé et de la précision que peut atteindre l'utilisateur, ainsi que du degré de couplage. Pour un suivi de doigts elle devra être millimétrique alors que pour les widgets virtuels (c.f. *Everywhere Display*, **S14** page 49) une précision centimétrique suffit. La stabilité statique est généralemet du même ordre que la précision. Pour les tâches de numérisation la précision à atteindre est limité par la résolution de l'oeil humain et par ça capacité à lire ou identifier l'information après numérisation. Une précision de l'ordre de 100 dpi est suffisante.

Une contrainte peu abordée par les auteurs qui décrivent les systèmes interactifs est l'autonomie. Nous en donnons la définition et la typologie suivante :

> L' **autonomie** d'un système désigne sa capacité à fonctionner avec ou sans assistance humaine ou logicielle, au moment de son initialisation (hors ligne) ou pendant son fonctionnement (en ligne).
>
> Nous qualifions un système perceptif de
>
> – *automatique* s'il fonctionne sans aucune assistance.
> – *coopérant* s'il requiert la coopération, ponctuelle ou continue, de son client applicatif, et/ou la coopération passive de l'utilisateur final.
> – *assisté* s'il requiert la coopération active de l'utilisateur final.

Le *client applicatif* est l'entité logicielle qui exploite le système perceptif (typiquement, une application interactive) : un système coopérant ou assisté interagit avec son client applicatif. Nous nommons *coopération passive* de l'utilisateur toute interaction qui interfère avec l'exécution de sa tâche, par opposition avec la *coopération active* qui l'interromp.

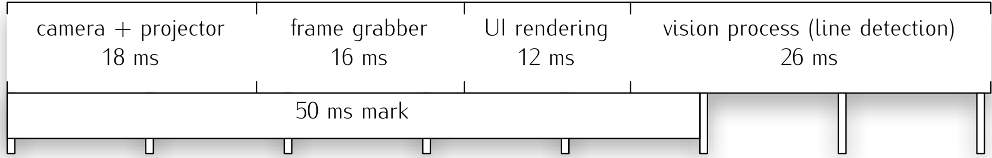


Figure 3.17 . Latence induite par les différents éléments d'un système interactif basé sur la vision par ordinateur.

Le matériel utilisé est : une caméra Sony EVID-30P couplée à une carte d'acquisition Bt878 ; une machine biprocesseur Pentium III à 1,8 GHz ; un projecteur video ESPON Tri-LCD.

Par exemple, la table *DiamondTouch* (utilisée dans plusieurs systèmes décrits plus haut) est coopérante en ligne : chaque utilisateur doit rester assis et n'utiliser qu'une main pour manipuler la surface. Le système utilisé dans *Head Gestures* (page 53) est assisté à l'initialisation (il requiert un long apprentissage auquel participe l'utilisateur) et automatique en ligne. Idéalement tout système perceptif devrait être automatique aussi bien à l'initialisation qu'en ligne. L'expérience montre que ceci n'est pas réaliste. De nombreux systèmes nécessitent au moins un calibrage à l'initialisation. C'est le cas, par exemple, de la table *DiamondTouch*. La plupart des systèmes perceptifs, en particulier ceux utilisant la vision, visent à être coopérant ou assisté à l'initialisation et automatique en fonctionnement.

La robustesse d'un système de perception est souvent critique ; c'est d'ailleurs principalement pour cette raison que la vision est peu employée en interaction Homme-machine. Pour clarifier, pour nous :

> Un système de perception est dit **robuste** si il maintient la qualité de service annoncée dans toutes les conditions possible de l'environnement pour lesquelles il est conçu.

La robustesse est donc une propriété binaire : soit le système reste dans les normes de qualité de service qu'il spécifie dans un environnement adapté, soit il n'est pas robuste. En outre, un système capable de détecter un environnement « hostile » (où il n'est pas capable de fonctionner correctement) et qui cesse d'y fonctionner peut être considéré comme robuste dans tout environnement.

3.3.1.2 Caractérisation de l'environnement

L'environnement de fonctionnement d'un système perceptif en vision par ordinateur n'est presque jamais décrit dans la littérature. Quand il l'est la description est succinte. Les experts de vision par ordinateur ne décrivent pas l'environnemt car ils le supposent constant ou avec des variations précises — par exemple uniquement des variations de luminosité. Les experts d'interaction Homme-machine ne le décrivent pas parce qu'il leur semble évident qu'un système perceptif puisse fonctionner dans toutes les conditions où ses utilisateurs finaux s'en servent. Or, connaître cet environnement est critique pour le choix des algorithmes de vision à employer. En outre, pour un système de vision donné l'environnement influe sur la qualité de service. Par exemple, l'implémentation de **S4'** utilisée pour la *Table Magique* cesse d'être utilisable si l'éclairage naturel est remplacé par un éclairage artificiel pendant l'utilisation.

En fait il n'existe pas à notre connaissance de manière formelle de décrire un tel environnment. Nous proposons la définition suivante de l'environnement d'utilisation d'un système perceptif visuel :

> L'environnement d'un système perceptif est l'ensemble des paramètres extérieurs au logiciel susceptible d'influer sur sa qualité de service.

L'environnement inclue en particulier : les propriétés du capteur utilisé, l'éclairage, et le degré de confusion de la scène (*clutter*).

Pour nous, l'environnement d'un système, ou l'environnement d'un besoin de perception, sera donc défini par

- la **caméra** et sa vue. Sont pertinents la définition de l'image fournie par le capteur (de 160x120 pixels, jusqu'aux hautes définitions atteintes par les appareils photo numériques) ; sa résolution sur la scène observés (les dimensions d'un objet de taille apparente de 1 pixel, en mètre/pixel) ; et la couleur du capteur : thermoscopie, infrarouge proche, visible en niveau de gris, ou visible en couleur (le plus courant). La mobilité ou non de la caméra (et donc de la vue) influe de manière importante sur la représentation du monde. Enfin, le cadrage est également pertinent. Il peut s'agir d'un cadrage « serré » qui inclue uniquement le lieu de l'interaction, d'un cadrage « large » qui inclue du contexte, ou d'un cadrage « quelconque » où la vue peut être fortement déformée par l'optique.

- l’ **éclairage** de la scène observé. Il est caractérisé par son degré d’uniformité et ses variations. L’uniformité peut être « uniforme » (cas de la lumière naturelle en exterieur), non-uniforme mais continu (cas d’une lampe éclairant une table, par exemple), ou discret (cas des persiennes éclairant une table). Les variations peuvent être constante et contrôlées ; constantes avec des fluctuations marginales (par exemple dues aux ombres projetées par les utilisateurs) ; variables avec des états discrets (par exemple selon les lampes allumées) ; ou variables continuement (c’est la cas de la lumière naturelle).
- la **confusion** de la scène. Ce critère est lié au cadrage de l’image caméra. Deux axes existent pour décrire la confusion de la scène : la complexité de l’arrière-plan et ses variations dans le temps. Pour la complexité, soit les agents d’interaction sont visibles seuls (sur un fond neutre uni par exemple), soit l’arrière plan contient d’autres objets. Les variations de l’arrière plan peuvent être constantes ou variables. Dans le cas variables, des objets parasites peuvent apparaître et disparaître de l’arrière plan. Ces objets parasites sont par exemple des passants (non-utilisateurs), ou des arbres, ou un écran video.

3.3.2 Taxonomie des services de perception visuelle

Nous disposons à présent de deux outils pour décrire un besoin de perception visuelle : une qualité de service à atteindre, et l’environnement dans lequel il est suceptible de l’atteindre. Ceci concerne la partie non fonctionnelle de la description d’un service.

Remarquons que les services rencontrés lors de l’état de l’art (récapitulés sur la table 3.16 page 55) s’intéressent à la perception d’un nombre limité d’agents que nous nommons *agents interactifs*. Il peut s’agir soit d’un objet plan, soit d’un corps humain ou d’une partie du corps. Ces services, indépendemment de l’agent étudié, fournissent une mesure d’une propriété des cet agent : sa localisation, son identité, son aspect, etc.

Ces considérations nous amènent à proposer la taxonomie suivante des services d’intérêt :

> On peut décrire un service de perception visuelle pour l’interaction par sa position dans une taxonomie comportant les quatre axes suivants :
> - l’ **agent interactif** d’intérêt du service
> - la **propriété** intrinsèque ou extrinsèque de cet agent qu’il mesure ou estime
> - la **qualité de service**, ou contraintes non fonctionnelles, qui s’appliquent pour cette mesure ou estimation
> - l’ **environnement** dans lequel le service est capable de maintenir cette qualité de service.

Cette taxonomie est représentée de manière graphique sur la figure 3.18 ci-contre.

Cette taxonomie n’est pas exhaustive. En particulier, notre définition de l’environnement d’un service est informelle et incomplète. Notre approche est pragmatique : nous proposons cette taxonomie comme un outil pour Laurence, afin de l’aider à exprimer ses besoins en terme de service de perception visuelle, et un outil pour Patrick ou Stan, afin de les aider à exprimer les services que leurs systèmes peuvent offrir.

3.3.2.1 Expression des besoins rencontrés

Chacun des services/besoins rencontrés au cours de ce chapitre peut être exprimé dans le cadre de la taxonomie proposée. Un rappel des définitions informelles de ces services est proposé sur le tableau 3.16 page 55.

Par volonté de concision nous ne donnons pas ici la représentation dans notre taxonomie de l’ensemble des services présentés dans ce chapitre. Cette ensemble fait l’objet de l’annexe C. Nous donnons toutefois un exemple : celui du service requis pour le

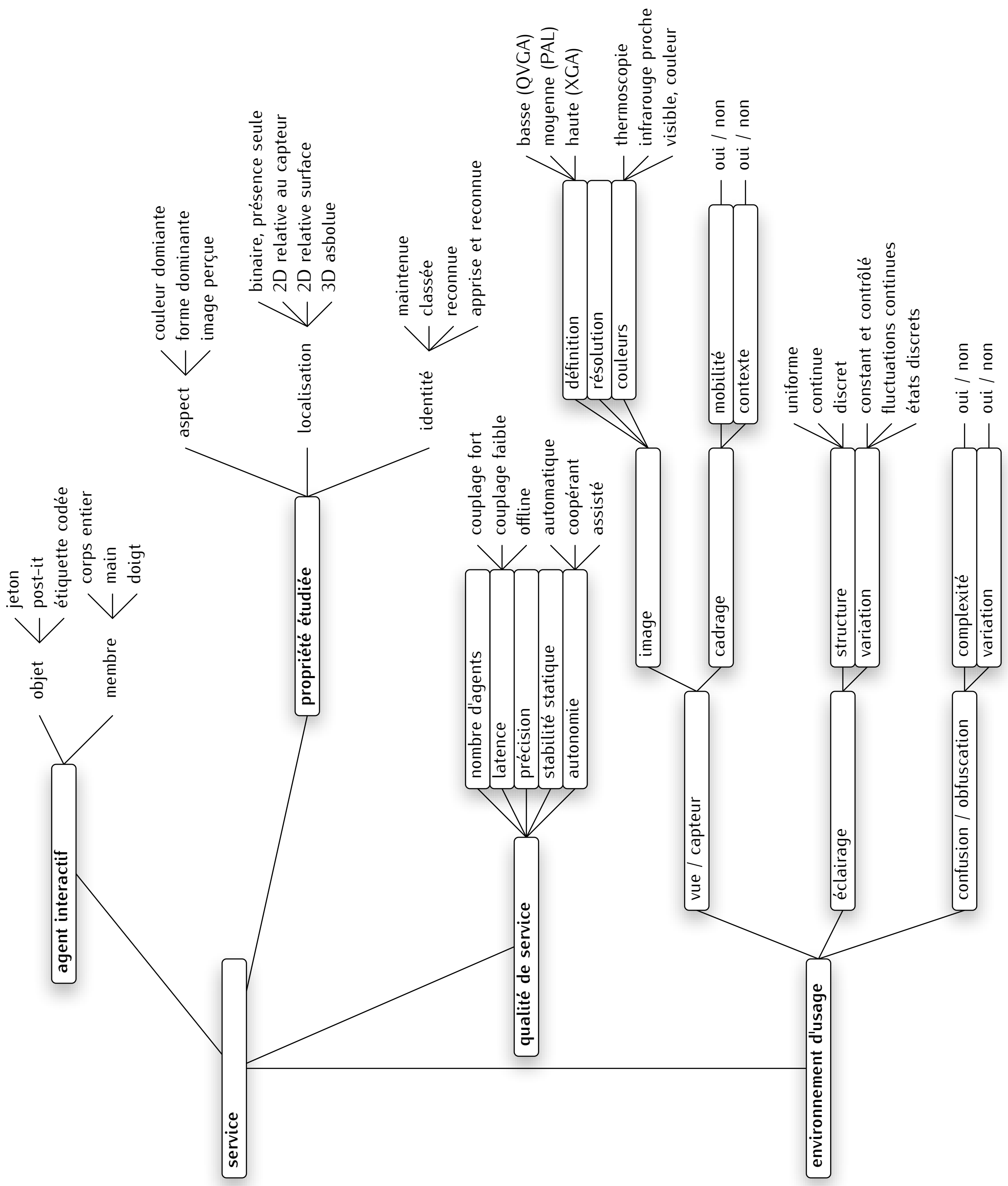


Figure 3.18 . Une taxonomie des services de perception visuelle pour l'interaction.

Les quatres noeuds fils de *service* représentent les quatre sous-espaces de la taxonomie, telle que définie ci-contre. Les noeuds encadrés n'ayant pas de descendants encadrés sont les axes de la taxonomie. Enfin, les noeuds sans cadres représentent les valeurs qui sont typiquement prises par un besoin ou un service selon un axe.

suivi des jetons de la *Table Magique*. Cette définition est la synthèse du besoin exprimé dans [Bérard, 2003] et résumé page 48. Ce besoin est exprimé d'une manière semi-graphique par le cartouche ci-après :

S4’						détection et suivi d'objets colorés de forme simple	
agent		*qualité de service*		*image & cadrage*		*eclairage*	
jeton rouge		#agents	0 à 10	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☒☐☐☐			contexte	☐	variation	☒

Une légende détaillée est fournie sur la figure 3.19 ci-contre. Ces cartouches seront utilisés dans ce manuscrit lorsqu'il sera nécesaire de décrire un besoin perceptif, ou un service perceptif.

Cette représentation répond à un besoin de concision et de lisibilité : une lecture suffit normalement à interprêter le besoin. Elle reste expressive pour les trois classes d'utilisateurs-développeurs : Stanislas, Patrick, et Laurence. Bien qu'informelle, nous pensons qu'il n'est pas possible de la détailler plus avant sans compromettre la propriété d'être une *lingua franca*.

Remarquons pour finir que cette représentation graphique est générée automatiquement à partir d'une représentation numérique.

3.3.2.2 Évolution de la taxonomie

Il est probable que notre analyse des besoins soit incomplète ou le devienne, car notre état de l'art peut être imparfait ; il ne peut de toute manière être valide qu'au moment de sa rédaction.

Laurence peut donc émettre un besoin qui n'est ni couvert par la boîte à outils que nous proposons dans les chapitres 5 et 6, ni représentable dans la taxonomie ci-avant ; cette dernière devra donc être complétée. Des spécialistes de vision (Patrick et Stanislas) devront alors couvrir le nouveau besoin. La taxonomie, ainsi que la démarche de conception que nous adoptons au chapitre 5 page 87, permettent même dans ce cas de faciliter la communication entre demandeur et fournisseur de service.

3.3.2.3 Justification du faible formalisme de la taxonomie

La taxonomie présentée ci-dessus est volontairement informelle. En effet, si la qualité de service requise peut être définie quantitativement pour chaque critère, et de manière compréhensible pour tous les acteurs, il n'en est pas de même pour les autres axes.

L'*agent interactif* pourrait par exemple faire l'objet d'une description plus formelle. Par exemple, un doigt (l'agent d'intérêt pour **S7** à **S10** en particulier) pourrait être décrit par ses dimensions, sa teinte, sa texture, ses poses possibles. Un jeton ou un post-it peut être décrit par sa teinte, sa forme géométrique. Mais une telle description devient dépendante de la vision. Pour la texture en particulier, une description numrique de la texture impose d'utiliser un descripteur d'aspect, par exemple un vecteur-réponse à des champs réceptifs gaussiens [Hall et al., 2000].

Une description numérique de l'*environnement* ne peut pas non plus être atteinte simplement. Par exemple, pour l'éclairage : il faudra décrire le ou les spectres de lumière tolérés, et dans quelles proportions ils le sont. Ceci dépendra également de la réponse spectrale du capteur utilisé.

Un description formelle ou quantitative qui peut être traitée par la machine n'est donc accessible qu'à des spécialistes en vision par ordinateur : la taxonomie perdrait son pouvoir expressif pour Laurence.

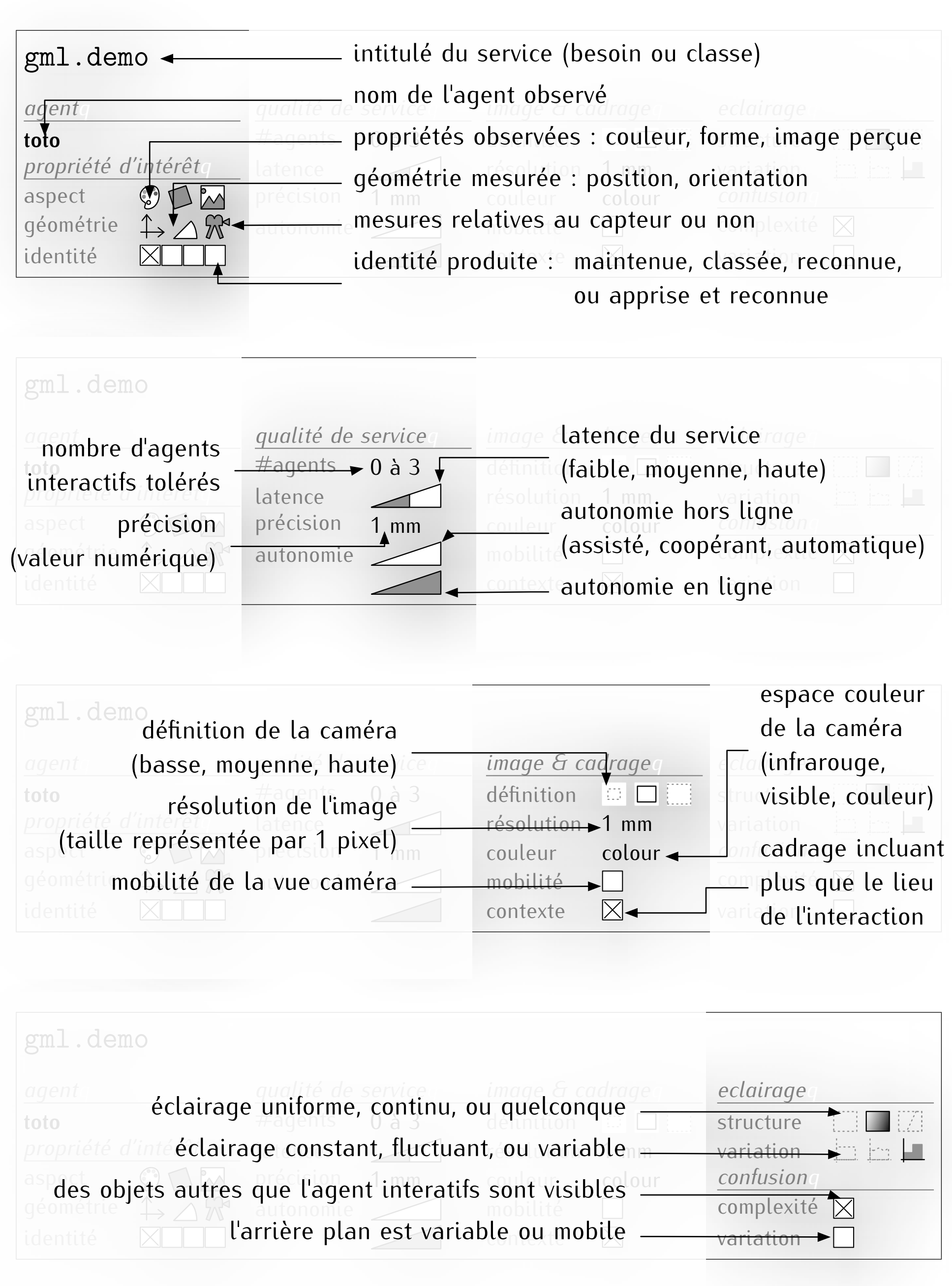


Figure 3.19 . Cartouches descriptifs d'un service ou d'un besoin.

Chaque élément fléché du cartouche correspond à un axe de la taxonomie proposée.

Enfin, si la taxonomie est trop complexe, elle perd son intérêt : (pouvoir) être lue et écrite par Laurence **et** par Patrick ou Stanislas, sans que l'effort dédié à sa manipulation devienne significatif par rapport à celui consacré au directement au développement et à l'évaluation du système interactif (pour Laurence) ou du service perceptif (pour Patrick et Stanislas).

3.3.2.4 Conclusions

L'objectif de cette taxonomie est d'être un outil de communication, principalement entre Laurence d'une part, et Patrick et Stanislas d'autre part. En décrivant un besoin comme volume suivant les quatre axes présentés Laurence peut l'exprimer de manière compréhensible pour Stanislas et Patrick. Réciproquement, en décrivant un service implémenté dans le cadre de cette taxonomie Stanislas et Patrick permettent à Laurence de choisir ou non de l'utiliser (ou de choisir parmi plusieurs services similaires) sur la base de son besoin de perception.

L'utilisation de cette taxonomie est donc une garantie d'**utilité** des services de perception visuelle pour l'interaction Homme-machine (les services et les besoins peuvent être exprimés dans le même langage, donc être compatibles) ainsi que de l'**utilisabilité** des services individuels (chaque service satisfait des contraintes non fonctionnelles précises).

Elle contribue donc à satisfaire les besoins d'utilité pour Laurence (concepteur de systèmes interactifs) et d'utilisabilité pour Caroline (utilisateur final des systèmes interactifs).

3.3.3 Conclusion

Dans ce chapitre nous avons complété la première partie de notre phase d'analyse : les besoins *fonctionnels* présents dans un ensemble de systèmes interactifs, que nous espérons représentatifs, ont été identifiés. Nous avons observé qu'ils possèdent des caractéristiques communes — ils s'interessent à une ou plusieurs propriétés d'une classe d'agents interactifs, ont des requis de performance spécifiques, et s'appliquent dans un environnement d'usage aux propriétés (plus ou moins) bien définies par les auteurs.

Pour que Laurence puisse décrire un besoin, et Patrick un service fourni, ils ont besoin d'un language commun. Nous avons donc synthétisé l'état de l'art pour construire une taxonomie des services perceptifs lisible par tous les acteurs du développement logiciel qui nous concernent.

La structure de cette taxonomie semble indiquer la posibilité de parvenir à un ensemble de service recouvrant les besoins de cardinal inférieur à celui des besoins exprimés. Nous présenterons au chapitre 5 une méthode pour atteindre ce recouvrement.

Cependant, si nous savons à présent *quels* services nous devons fournir pour rendre une boîte à outils utile, nous devons déterminer *comment* les présenter à Laurence et permettre à Patrick de les implémenter pour réaliser une boîte à outils utilisable.

4 Requis structurels pour une boîte à outils utile et utilisable

« (...) The software architect needs a number of different views of the software architecture for the various uses and users. At present we make do with only one view : the implementation. In a real sense, the implementation is like a builder's detailed view — that is, like a building with no skin in which all the details are visible. It is very difficult to abstract the design and architecture of the system from all the details. »

[Perry et Wolf, 1992]

Dans le chapitre précédent, nous avons traité de l'utilité et de l'utilisabilité d'une boîte à outils de vision par ordinateur pour l'interaction Homme-machine, pour certains des utilisateurs cible : Caroline et Laurence. Nous avons proposé un ensemble de requis concernant le « fond », c'est-à-dire quels services fournir dans une boîte à outils.

Dans ce chapitre, nous désirons obtenir un ensemble de requis sur la « forme », c'est-à-dire *comment* fournir les services nécessaires. Ils sont imposés par les besoins des utilisateurs directs de la boîte à outils ; par ordre de priorité : Laurence, qui l'utilise pour développer des applications ; Patrick, qui l'utilise pour intégrer des services ; enfin Stanislas, qui implémente des algorithmes de perception.

Pour Laurence, il s'agira de déterminer comment une application interactive se « connecte » à la boîte à outils et communique avec elle. Pour Patrick, nous déterminerons comment les différents morceaux de la boîte à outils — pour ne pas utiliser le terme de *composants*, trop chargé de sens — peuvent être liés les uns aux autres, déclenchés, ou contrôlés, pour fournir des services plus adaptés à une tâche particulière ou des services fonctionnellement nouveaux. Enfin, pour Stanislas, nous étudierons comment fournir l'infrastructure qui permet « d'emballer » des algorithmes de vision sous une forme adaptée à Laurence et Patrick.

Après avoir présenté la notion de service, et la notion d'architecture logicielle, nous présentons des critères permettant d'évaluer des boîtes à outils et leurs architectures (section 4.1). Nous utilisons ensuite ces critères pour analyser des boîtes à outils existantes (section 4.2), et en tirer un ensemble de requis, ainsi que des choix architecturaux. La plupart des boîte à outils étudiées sont des bibliothèques de vision pour l'interaction, et certaines de domaines connexes.

Pour synthétiser ce second état de l'art, nous détaillons (section 4.3) les requis centrés utilisateur en requis non fonctionnels (latence, autonomie, et fiabilité de l'architecture concrète) et en requis fonctionnels (asynchronisme, abstraction, isolation, et contrat). Ils nous guident vers l'approche logicielle orientée service que nous présentons au chapitre suivant.

À la fin de ce chapitre, nous aurons donc déterminé l'ensemble des besoins que notre boîte à outils, *gmlVision*, doit satisfaire. Nous devrons alors, dans le chapitre suivant, proposer une approche de conception et une architecture adaptée.

Le chapitre est organisé comme suit : nous donnons un ensemble de définitions qui nous permettent de décrire la structure d'une boîte à outil et de l'évaluer. Nous passons ensuite en revue les différentes approches possibles en les évaluant vis à vis des besoins de nos utilisateurs. Nous effectuons alors une synthèse de cette évaluation afin de choisir l'approche que nous retenons pour notre boîte à outil.

4.1 Définitions

Pour définir les critères du succès d'une boîte à outils vous utilisons les termes « service » et « architecture ». Ces termes étant fréquemment utilisés dans différents contextes où ils désignent des concepts sensiblement différents, nous prenons soin de préciser leur sens dans notre discours.

4.1.1 Service et architecture

Selon la communauté de recherche en informatique, le terme *service* peut avoir des sens différents. Sassne et Macmillan en donnent une définition très générale :

> Un service effectue un travail utile sans produire de bien tangible, (...) au travers d'une interface bien définie, qui permet en particulier sa composition avec d'autres services. [Sassen et Macmillan, 2005]

Notre vision d'un service logiciel est donnée par la définition suivante, adaptée de [MacKenzie et al., 2006]) :

> Un service [logiciel] est la combinaison des concepts suivants :
> – la capacité à accomplir une tâche pour un autre ;
> – la spécification de la fonction et de la qualité du travail fourni ;
> – l'offre de l'accomplissement de la tâche.

Dans le cadre de la vision pour l'interaction, un service perceptuel pour l'interaction détermine les actions d'un utilisateur en fonction de données issues d'un capteur.

En pratique, un service une l'entité logicielle qui implémente un besoin décrit dans le cadre de la taxonomie proposée page 58. Rappelons que cette taxonomie permet aussi bien d'exprimer le *besoin* de Laurence ou Patrick que le *service*, réponse au besoin, implémenté par Patrick ou Stanislas.

Structurer notre boîte à outils pour pemettre à Patrick de construire un service, et à Laurence d'exploiter l'ensemble des services fournis, correspond à définir une architecure. La définition suivante de l'architecture [Bass et al., 2003] semble faire référence :

> « L'architecture logicielle d'un programme ou d'un système informatique est la structure (...) du système ; elle comprend des éléments logiciels, leur propriétés visibles, et leurs relation et procédés de mise en relation.
> Le terme *visible* désigne les suppositions qu'un élément peut formuler sur un autre, comme les services fournis, les caractéristiques de performances, l'utilisation de ressources, etc. »

Pour structurer un ensemble de services sous la forme d'une boîte à outils, il est nécessaire de décider d'une interface utilisateur externe (pour Laurence), d'une interface utilisateur interne (pour Patrick et Stanislas). Ceci correspond aux « propriétés visibles » des éléments. Il est également nécessaide de définir une manière de permettre la communication entre les différentes composantes de la boîte à outils. C'est l'ensemble des ces décisions que nous nommons *architecture abstraire* de la boîte à outils.

L'*architecture concrète* de notre future boîte à outils *gmlVision* résulte ensuite des choix technologiques implémentant ces décisions.

Dans la suite de ce chapitre, nous nous attachons à décrire et à évaluer les différentes approches architecturales utilisées pour des boîte à outils de services de vision pour l'interaction, vis-à-vis de critères centrés sur les besoins de nos utilisateurs.

4.1.2 Critères d'évaluation des approches

À notre connaissance il n'existe pas de travaux sur l'evaluation d'une API (*application programmer interface*) en tant qu'interface utilisateur. Dans notre cas l'utilisateur à satisfaire est Laurence (qui conçoit des systèmes interactifs utilisant la boîte à outils), et en second lieu Patrick et Stanislas (qui développent et étendent la boîte à outils). Comment déterminer si une boîte à outils est satisfaisante, et si son approche architecturale est valide ?

Myers fournit des éléments pour évaluer ce type d'outils [Myers et al., 2000]. Bien qu'utilisés essentiellement dans le cas des boîtes à outils de construction d'interfaces graphiques, nous pensons que ces considérations sont également valables pour nous. Nous les rappelons ici, en montrant comment elles sont applicables et en les rattachant aux notions d'utilité et d'utilisabilité.

4.1.2.1 Choix de la cible

D'après Myers, les « outils qui ont réussi sont ceux qui aident [le développeur] juste où cela est nécessaire. » En d'autre termes, il faut « viser juste », et éviter la sur-utilité ou *overshot*.

Nous interprétons l'objectif de viser juste à deux niveaux : au niveau le plus général, l'objectif fondamental de notre boîte à outil est de fournir les services de perception visuelle nécessaires au développeur de système interactif. Il pourrait être tentant d'étendre les services de notre boîte à outil pour couvrir des besoins connexes aux besoins de perception identifiés. Par exemple nous avons identifié le service **S14** « détection de l'occlusion d'un objet » au chapitre précédent page 49. Il pourrait sembler naturel de fournir un atelier de construction d'interfaces basées sur les boutons sensibles aux occlusion (Interface Builder). Mais ce serait s'écarter de notre objectif fondamental. Concevoir un atelier de construction d'interfaces est un problème complexe traité par ailleurs. Le critère de viser juste nous indique qu'il est préférable de fournir un service de perception qu'il sera facile de mettre en oeuvre (l'objet de ce chapitre) dans un atelier de construction existant.

L'objectif de « viser juste » s'applique également à un niveau plus fin : celui de l'étendu des services de perception que nous cherchons à couvrir. Ici encore, il peut être tentant d'étendre la couverture d'un besoin identifé afin de prévoir d'autre besoins potentiels. En reprenant l'exemple **S14**, nous pourrions étendre le service pour qu'il fournisse un indice d'occlusion à plusieurs niveaux et pas simplement binaire. Mais ce choix aurait sans doute des conséquences négatives sur la robustesse du service alors que rien n'indique que le nouveau service couvrira un jour un besoin réel. Notre boîte à outil doit donc viser juste en se limitant aux services pour lequels nous avons montré un besoin réel dans notre état de l'art au chapitre précédent.

Cet objectis s'applique, enfin, à la satisfaction des besoins des autres utilisateurs (Patrick et Stanislas). Nous devrons leur offrir une API précisément adaptée à leurs besoins. Comme cette cible est secondaire, ceci ne fait pas ici l'object d'une étude approfondie. Nous y reviendrons en décrivant l'approche adoptée pour l'architecture de la boîte à outils, section 5.2 page 93.

4.1.2.2 Seuil et plafond fonctionnels

Le *seuil* (ou plancher) d'une boîte à outils représente l'effort nécessaire à un développeur pour apprendre à l'utiliser et commencer à en bénéficier. Le « plafond » représente jusqu'où peut aller la boîte à outils. C'est en fait la limite entre l'ensemble des systèmes qu'il est possible de réaliser avec la boîte à outils et l'ensemble de ceux qui ne pourront pas bénéficier de la boîte à outil du fait de ses limitation fonctionnelles.

Un objectif lors de la conception d'une bibliothèque logicielle est bien entendu de minimiser le « seuil » et de maximiser le « plafond » : on obtient ainsi une bibliothèque facile à apprendre, et peu limitative. Quelques propriétés permettant d'atteindre cet objectif sont proposés par [Klemmer et al., 2004] :

– la facilité d'utilisation : inclut en particulier le temps d'apprentissage pour les novices, et la lisibilité des programmes utilisant la boîte à outils pour d'autres développeurs ;
– la facilité de réutilisation : un outil de développement devrait permettre la réutilisation du code produit pour des problèmes comparables, afin de minimiser l'effort de développement ;
– les mêmes patrons produisent le même code : afin de minimiser les erreurs de conception, il est souhaitable que la structure de la boîte à outils incite les programmeurs à utiliser des patrons de conception particuliers, et de minimiser l'effort à fournir lorsqu'ils sont utilisés.

Rappelons que pour notre boîte à outils, la cible principale est Laurence. Pour que le **seuil** soit bas, il convient de permettre qu'un service perceptif soit instanciable en en minimum de lignes de code. En plus, la manière d'instancier un service doit être suffisemment compréhensible (par exemple, proche de son expression dans la taxonomie) pour que l'effort d'apprentissage soit minimal. Élever le **plafond** est plus complexe : il faudrait garantir que la boîte à outils permet de réaliser des systèmes que nous n'avons pas pris en compte dans l'état de l'art — c'est-à-dire, de rendre la boîte à outils générique.

Pour les deux autres cibles (Patrick et Stanislas), les implications sont différentes. Il est nécessaire de proposer des mécanismes simples pour prototyper des services perceptifs — c'est-à-dire, les construire et expériementer avec ces services, en fournissant un minimum d'efforts. Les « briques élémentaires » de la vision par ordinateur, algorithmes de traitement d'image en particulier, devront être fournies et aisément utilisables. En particulier, elles doivent autant que possible fonctionner sans configuration : en d'autres termes, posséder une configuration par défaut correspondant à une situation classique. Les « briques » plus complexes, construites par Stanislas, devront pouvoir être réutilisées : la boîte à outils devra fournir des mécanismes de réutilisation de code à Patrick et à Stanislas.

4.1.2.3 Chemin de moindre résistance

En paraphrasant Myers : l'API d'une boîte à outils influence la structure des services qu'elle permet de construire. Si elle est bien conçue, elle peut les rendre plus utilisables. Ce critère est semblable au troisième critère de Klemmer présenté dans les paragraphes précédents.

Ceci s'applique en particulier Stanislas et Patrick. La manière la plus évidente de construire un service, ou d'assembler des « briques » élémentaires pour construire un algorithme de traitement d'images, doit être la « bonne » ; c'est-à-dire celle qui produit un service utilisable.

D'autre part, les développeurs ne doivent pas avoir besoin, en général, de contourner les structures en place (qui correspondent à un patron de conception choisi par l'architecte de la boîte à outils) pour accomplir une tâche particulière. Il convient donc de fournir une API répondant aux besoins typiques des clients (Laurence, Patrick et Stanislas) plutôt qu'une API générique. Nous verrons au chapitre suivant que le choix d'une API spécifique plutôt que générique est, par nature de la taxonomie des services proposée, d'autant plus pertinent pour la principale tâche de conception de Laurence (le choix d'une instance de service perceptif).

En ce qui concerne Laurence, notons qu'offrir un chemin de moindre résistance consisterait à l'inciter à faire les bons choix de conception pour son système interactif. Ceci n'est cependant pas possible pour les interfaces perceptuelles : c'est un domaine émergent, où il n'existe pas encore de conventions ou de « bonnes pratiques » identifiées. Par contre, il en existe pour les interfaces graphiques (GUI) classiques, avec lesquelles Laurence est familère. Nous pourrons nous appuyer sur ces conventions pour définir l'interface de notre boîte à outils.

4.1.2.4 Prévisibilité

« Les outils qui utilisent des techniques automatiques parfois peu prévisibles sont mal reçus des programmeurs. » Ceci a des implications profondes pour l'aspect fonctionnel de certains services perceptifs. En effet, par nature, l'environnement d'usage est peu prévisible.

Dans notre cas, nous pouvons choisir d'implémenter des services qui s'arrètent *automatiquement* de fonctionner lorsque les conditions extérieures (l'environnement) cessent d'être compatible avec une qualité de service acceptable : ceci les rend plus robustes.

Du point de vue de Caroline, ceci est problématique : elle peut être interrompue dans sa tâche sans notification ni explication.

Si de tels comportements automatiques sont mis en place, il faudra donc fournir des mécanismes d'*observabilité* aux clients pour compenser la perte de prévisibilité ; il pourront alors notifier leur propre client, en cascade jusqu'à l'utilisateur final. Reprenons l'exemple ci-dessus :

- Stanislas et Patrick doivent avoir la possibilité d'inspecter l'état d'un service et de ses composants, en ligne, afin de choisir ou nom d'interrompre la perception ;
- Laurence doit être notifiée de l'état de « santé » d'un service, ainsi que d'un éventuel diagnostic en cas d'échec (par ex. « trop d'objets parasites », « éclairage modifié ») ;
- Caroline doit avoir le moyen de connaître cet état de santé et, le cas échéant, comment remédier au problème.

4.1.2.5 Cibles mobiles

Lorsqu'on construit un outil innovant, il est difficile de bien viser. Autrement formulé, il est difficile d'innover avant d'avoir « suffisemment d'expérience et de compréhension des tâches supportées [par l'outil]. » Toujours d'après Myers, au moment où la tâche devient assez bien connue, elle peut être devenue moins importante ou même obsolète. Il y a également un risque que l'usage effectif de l'outil soit très différent de celui prévu lors de sa conception.

Notre objectif n'est pas de faire progresser le domaine de la vision en soi. Il est prévisible que de nouvelles solutions en vision rende caduque les solutions employées dans botre boîte à outils. Mais à défaut d'être utilisables par Laurence, ces nouvelles solution ne rendront pas caduque notre boîte à outils.

Par ailleurs, du point de vue de l'interaction Homme-machine, nous considérons que le risque de la cible mobile est limité du fait de l'effort de recherche important qu'il reste à produire pour que la conception d'interactions perceptive soit maîtrisée et standardisée.

4.2 Familles d'approches existantes

Pour construire une boîte à outils de services logiciels, nous identifions quatre familles d'approches : les bibliothèques « classiques » rendues populaires par les premiers UNIX ; les approches monolithiques ; les approches à composants ; et les approches à services.

Nous décrivons brièvement chacune des ces approches ci-après, en les évaluant informellement vis-à-vis des requis évoqués plus haut. Pour chaque approche, nous présentons des boîtes à outils correspondantes, destinées à la perception pour l'interaction, et s'adressant epxlicitement à au moins l'un de nos utilisateurs directs (Laurence, Patrick ou Stanislas). Nous résumons également pour chaque bôite à outils présentée les éventuelles conclusions des concepteurs de la boîte à outils sur les requis centrés utilisateur. Cette inspection nous permettra de conclure sur le choix d'une approche (l'approche orientée service) pour la conception de notre boîte à outils.

4.2.1 Bibliothèques classiques

Cette catégorie contient les bibliothèques logicielles au sens d'UNIX ou du langage C. Il s'agit de collections de sous-programmes utilisés pour développer des logiciels. Une telle bibliothèque contient du code et des données, qui fournissent des services pour la construction de programmes indépendants. Ceci permet l'échange (entre développeurs) et le partage (entre applications) de code et de données de manière modulaire. Le code est *lié* à une application de manière statique ou dynamique. L'application accède à la fonctionnalité de la bibliothèque par l'appel de procédures ou la création d'objets.

Ces bibliothèques se contentent d'étendre un langage en ajoutant à sa bibliothèque standard de nouvelles fonctionnalités. Il peut s'agir d'un langage compilé (typiquement C ou C++) ou d'un langage dynamique ou interprété (typiquement Java, Python ou Tcl).

4.2.1.1 OpenCV

OpenCV [Bradski et Pisarevsky, 2000], est un exemple assez récent de bibliothèque de vision par ordinateur, et largement déployé dans notre communauté scientifique. L'objectif principal de cette bibliothèque est de fournir des services de perception pour la construction d'applications interactives. Les fonctionnalités fournies sont *a priori* très attirantes pour Laurence : identification et reconnaissance d'objets, détection et reconnaissance de visages, reconnaissance de gestes, suivi de mouvements.

Cependant la bibliothèque n'est pas utilisable par Laurence. Prenons pour exemple l'accès à la fonctionnalité de reconnaissance de visages, qui est l'un des points forts de la bibliothèque. Il s'agit du premier exemple non trivial fourni par la documentation de la bibliothèque [Bradski, 2006]. Pour ce faire, il est nécessaire de comprendre la notion de *Haar-like Features* (la technologie employée pour la détection), et d'écrire environ 250 lignes de code en langage C qui manipulent (en particulier) ce concept. Remarquons en outre qu'aucune information n'est disponible sur la qualité de service atteinte. Ici, l'API est de trop bas niveau d'abstraction : le ciblage est mauvais, et le seuil fonctionnel trop élevé.

Un autre défaut d'utilisabilité est dû au manque de cohérence de la structure et de l'API de la bibliothèque. Plusieurs services logiciels fonctionnellement similaires y sont présents, sans qu'il soit possible à l'utilisateur non expert de discriminer entre eux. Supposons que Stanislas utilise OpenCV. La tranformée de Hough [Duda et Hart, 1972] est un outil classique de vision qui permet de détecter des lignes droites dans une image ; il existe dans OpenCV trois variantes, qui

– sont adaptées à des conditions différentes : en particulier, à des environnements plus ou moins confus ;
– ont des performances différant de deux ordres de grandeur : l'une est utilisable pour des applications « temps réel » (i.e. pour des interactions fortement couplées), les deux autres non ;
– ont des API légèrement différentes : les paramètres d'initialisation ne sont pas les mêmes.

Aucun indice ne permet de sélectionner l'une des trois variantes, en fonction de la qualité de service requise. Comme elles n'ont en outre pas la même API, il est malaisé pour Stanislas de les comparer. C'est également le cas d'autres algorithmes dans OpenCV (calcul du flot optique, détection de bords ou de coins, etc).

En conclusion, OpenCV a des défauts d'utilisabilité pour Laurence et pour Stanislas. Nous attribuons son succès au fait que, cependant, cette bibliothèque logicielle rassemble des implémentation classiques ou performantes de nombreux algorithmes.

4.2.1.2 Papier-Mâché

Cette bibliothèque [Klemmer et al., 2004] est déstinée à la construction d'interfaces tangibles (c.f. section 3.2.1 page 40). Son objectif est de rendre accessible à Laurence l'expérimentation avec des interfaces tangibles, i.e. de baisser le seuil fonctionnel.

Les auteurs ont choisi de mener une étude de terrain préalable à la conception de la boîte à outils *Papier-Mâché*. Ils ont interrogé des chercheurs ayant bâti des interfaces tangibles afin d'identifier leurs besoins fonctionnels et non fonctionnels. Nous résumons ici celles de leurs conclusions qui nous concernent, c'est-à-dire celles qui ne sont pas spécifiques aux interfaces tangibles. Elles sont centrées sur l'abaissement du seuil de la bibliothèque :

- un service perceptif doit fournir des informations abstraites en termes d'objets (ou d'agents d'interaction), pas en termes de pixels ou d'images ;
- l'abstraction appropriée pour transmettre les informations perçues est le formalisme à événements, comme dans les boîtes à outils traditionnelles de construction d'interfaces WIMP ;
- les événements doivent être uniformes pour différentes technologies des perception [fournissant le même service], afin de pouvoir substituer une technologie à une autre facilement ;
- il faut fournir du *feedback* à Caroline, c'est-à-dire rendre observable le (bon) fonctionnement de la perception par du retour visuel ;
- la boîte à outils doit également fournir à Laurence des moyens de visualiser le fonctionnement du service perceptif, à des fins de débogage
- il est nécessaire de pouvoir simuler la perception, c'est-à-dire générer des événements envoyés à l'application (technique dit du *magicien d'Oz*).

L'évaluation de Papier-Mâché présentée par ses auteurs montre que la bibliothèque parvient à son objectif : son seuil est très bas, et elle permet de construire des applications interactives très rapidement. Cependant, comme OpenCV, Papier-Mâché ne fournit que peu d'information sur la qualité de service atteint par ses services perceptifs. Les auteurs ont mesuré approximativement la latence des services fournis (comparables à **S4**, c.f. page 43), et admettent qu'elle ne satisfait pas entièrement le besoin ciblé (ils ne sont pas utilisables pour l'interaction fortement couplée).

Enfin, Papier-Mâché impose à Laurence d'utiliser l'environnement Java/Eclipse pour développer son application interactive. Ceci contribue à élever son seuil, puisque Laurence devra faire un effort d'interopération avec cet environnement si le sien est différent.

4.2.1.3 Conclusions sur les bibliothèques classiques

L'observation de OpenCV et *Papier-Mâché*, ainsi que d'autres bibliothèques que nous qualifions de « classiques » nous permettent de formuler plusieurs remarques et critiques.

Ces deux boîtes à outils, ne semblent pas atteindre le bon niveau d'abstraction. OpenCV fournit des services qui ciblent souvent sans distinction Laurence, Patrick, et Stanislas ; Papier-Mâché cible Laurence exclusivement, au détriment de son extensibilité (donc de son plafond). D'autre part, les problèmes de qualité de service ne sont exprimés ni de manière logicielle, ni *via* une documentation : aucune des deux bibliothèques ne précise les performances ou l'environnement d'usage des services qu'elle propose. Ceci est également un défaut de conception plutôt que d'architecture.

Plus généralement, la portabilité et l'interopérabilité des services n'est pas traitée, par nature de ces bibliothèques : l'utilisateur est verouillé dans un environnement logiciel ou matériel. Hors Laurence est suceptible d'utiliser des plates-formes matérielles et logicielles variées (typiquement Java, Tcl/Tk, Python), et généralement de plus haut niveau que les langages compilés de type C (utilisés dans ces bibliothèques).

4.2.2 Approches monolithiques

Certaines bibliothèques ou parties de bibliothèques ont une structure que nous qualifions de *monolithique* (en anglais, le terme *framework* est souvent employé). Un monolithe est un squelette de programme, dont on définit la fonctionnalité en définissant des procédures ou méthodes particulières, appelée *hooks*. Typiquement, il s'agit de définir une procédure d'initialisation qui sera exécutée au début du programme, des procédures de réponse aux événements utilisateur ou internes, etc.

Par exemple, la bibliothèque GLUT (*OpenGL Utility Toolkit* [Kilgard, 1996]) fournit un monolithe pour la construction d'applications interactives dont l'affichage se fait via *OpenGL*. Pour l'utiliser, il suffit au développeur d'implémenter

- un *hook* qui sera appelé chaque fois qu'il est nécessaire de mettre à jour l'affichage de l'interface (`glutDisplayFunc`) ;
- un *hook* pour chacun des événements utilisateur (touche clavier, souris, ou joystick) auquel le développeur souhaite voir réagir l'application (`glutKeyboardFunc`).
- un *hook* qui sera appelé que du temps processeur est disponible pour des traitements annexes (`glutIdleFunc`) ;
- enfin, une procédure d'entrée (`main`) qui initialise la bibliothèque et déclare les *hooks*.

4.2.2.1 ImaLab

ImaLab [Lux, 2004] est un environnement de conception d'applications interactives, utilisé pour la recherche en vision par ordinateur. Il exploite pour ce faire la bibliothèque (classique) PrimaVision. Il a été conçu dans l'équipe PRIMA et a été déployé de manière confidentielle dans plusieurs autres équipes de recherche et dans un cadre industriel.

Implémenter un système perceptif avec ImaLab consiste à écrire une procédure de traitement d'image, correspondant au *hook* d'affichage de GLUT. Elle est chargée d'extraire l'information de perception de l'image courante à l'aide des primitives de PrimaVision, éventuellement en utilisant ou en mettant à jour un état interne. Cette procédure peut également produire un affichage graphique, à des fins de monitoring ou de débogage. Une fois cette procédure principale écrite, ImaLab permet de « lancer » la perception en lui passant chaque image issue d'un flux video.

Les concepteurs de cette boîte à outils ont abaissé son seuil fonctionnel (pour Stanislas) en fournissant certaines fonctionnalités, qui en font un monolithe. L'objectif est de permettre le prototypage rapide de services perceptifs :

- l'accès à l'image courante de la caméra est simplifié : elle est toujours nommée `current-image` dans le shell ImaLab ;
- l'affichage graphique des résultats de traitement est automatique. Il est possible à Stanislas de naviguer dans l'historique des traitements ;
- le monolithe fournit la possibilité de basculer d'un mode « temps réel » à un mode « pas à pas ».

Ces points sont des avantages pour Stanislas, mais nous considérons qu'ils constituent également des limitations de la boîte à outils. ImaLab est principalement une victime de l'effet de « cible mobile » décrit par Myers : les besoins au moment de sa conception ne sont plus les mêmes que les besoins actuels. Par exemple, il est délicat, dans cette boîte à outils, de traiter plusieurs sources video simultanées ; il est nécessaire d'exécuter plusieurs monolithes et de les faire communiquer *via* un mécanisme d'IPC (*Inter Process Communication*). Pour Patrick, cela signifie qu'il est plus difficile d'intégrer un système perceptif utilisant plusieurs caméras.

La boîte à outils utilise par ailleurs un langage devenu inhabituel (une variante de Scheme), et un langage *ad hoc* (un langage interprété dérivé de C++). De plus les primitives de traitement d'image fournies sont souvent redondantes. Ceci contribue à augmenter le seuil d'utilisabilité pour Stanislas et pour Patrick.

Enfin, ImaLab ne traite pas les besoins de Laurence : en tant que monolithe, le système perceptif et l'application interactive ne sont pas indépendants, et les notions manipulées ne sont pas utilisables pour Laurence. Le modèle de développement est *ad hoc*, c'est-à-dire que système perceptif et l'application interactive sont conçus et développés d'un bloc, par les mêmes utilisateurs, en pratique correspondant au profil de Stanislas.

4.2.2.2 Augmented Reality Toolkit

Cette boîte à outils [Kato et al., 2000] s'approche particulièrement du paradigme de d'application monolithique, GLUT. Il s'agit d'ailleurs d'une extension de GLUT. Son objectif est de permettre la construction d'application en réalité augmentée (RA), en détectant des marqueurs connus placés dans l'environnement ou sur des objets. Elle fournit au client les coordonnées des marqueurs détectés dans le réferentiel de la caméra.

Les *hooks* qu'il est nécessaire d'implémenter une application sont les suivants :

- `init` initialise l'application (en particulier, l'accès à la caméra et à GLUT).
- `mainLoop` représente la boucle d'événements ; cette procédure est appelée par le monolithe lorsque la caméra fournit une nouvelle image. L'utilisateur doit y appeler une procédure d'acquisition video, une procédure de détection de marqueurs, et une procédure de calcul de transformation géométrique.
- `draw` est chargée du rendu graphique de la scène augmentée.
- `cleanup` libère les ressources utilisées.

La bibliothèque ARToolkit est copieusement documentée, et la simplicité de la structure permet de créer des systèmes interactifs en RA très rapidement : le seuil est bas, et le chemin de moindre résistance bien tracé. Néanmoins, cette boîte à outils pêche par deux points importants :

- le niveau d'abstraction n'est pas adapté à Laurence, pourtant cible principale de la bibliothèque. Par exemple, elle doit manipuler directement les images de la caméra ; fournir une information de seuillage pour la détection des marqueurs. Caroline, elle, doit effectuer une opération de calibrage géométrique non trivale au déploiement de l'application interactive.
- la qualité de service n'est pas spécifiée. Dans un environnement considéré comme « normal » pour Laurence, par exemple un bureau avec un éclairage mixte entre lumière naturelle et lumière artificielle, elle est d'ailleurs inutilisable. Les oscillations de suivi observées sont centimétrique (sur un marqueur de l'ordre de 5 cm), et les resultats sont occasionnellement abberrants. Laurence doit donc travailler en environnement contrôlé ou réaliser un post-traitement sur les résultats.
- le plafond est trop bas. ARToolkit limite Laurence aux applications de RA où des images virtuelles sont superposées à la vue de Caroline de la scène réelle (typiquement *via* un casque opaque ou semi-transparent). Les autres applications de la RA, par exemple les surfaces interactives, sont difficiles à implémenter à l'aide de cette bibliothèque.

Pour Patrick et Stanislas, c'est la structure de la bibliothèque qui est problématique. Come tout monolithe, elle intègre la capture video, le traitement en vision par ordinateur, l'application interactive, et le retour visuel. Il est donc difficile de l'assembler avec d'autres systèmes perceptifs, ou d'utiliser une autre caméra. Un autre groupe de chercheurs [Wagner et Schmalstieg, 2006] propose d'ailleurs, afin de remédier aux défauts de l'ARToolkit, une boîte à outils simplifiée (nommée *ARToolkitPlus*), qui

- ne contient que les fonctionnalités de perception : c'est une bibliothèque classique, plus un monolithe ; les composants d'acquisition video et de rendu ont été supprimés.
- améliore et specifie la qualité du service perceptif, en termes de précision et de stabilité statique.

4.2.2.3 Conclusions sur les bibliothèques monolithiques

Comme les bibliothèques classiques, les monolithes verrouillent Laurence dans un environnement de développement particulier, potentiellement différent ou incompatible avec celui qu'elle a choisi au moment de la conception de son application interactive. Ils permettent d'atteindre un seuil très bas en minimisant l'effort de développement, et en fournissant un chemin de résistance faible, mais le plafond fonctionnel s'en trouve abaissé. En particulier, il devient difficile de réutiliser un service perceptif, que ce soit à la conception d'un nouveau système interactif ou lors de l'utilisation de plusieurs systèmes devant utiliser simultanément le même service.

4.2.3 Approches à composants

Un composant logiciel est un élément offrant un service défini par son concepteur, et capable de communiquer avec d'autres composants. D'après [Messerschmitt et Szyperski, 2003], un composant doit remplir les conditions suivantes :

- être réutilisable et utilisable en parallèle par plusieurs clients ;
- ne pas dépendre du contexte, c'est-à-dire de l'état du reste du système ;
- pouvoir être composé (assemblé) avec d'autres composants ;
- être encapsulé, i.e. ne pas posséder d'état interne observable à travers son interface — en d'autres termes, avoir une interface purement fonctionnelle ;
- être indépendant, du déploiement du reste du système, et des versions (des autres composants).

Un composant se conforme à une spécification, généralement fortement liée à un environnement de développement, à un langage de programmation, ou à un système d'exploitation spécifique : COM, Microsoft .NET, Java Beans. Cette spécification garantit, en particulier, sa réutilisabilité. Son interface est souvent décrite dans un langage de description d'interface (*Interface Description Language*, par exemple DCE/RPC IDL dans COM). Cette description, une fois compilée, fournit des adaptateurs ou proxy (au sens des patrons de conception classiques, voir [Gamma et al., 1995]) qui permet à un composant d'exister de manière autonome d'autres ou d'applications dans un ordinateur.

4.2.3.1 Vista

Vista [Pope et Lowe, 1994] est l'une des premières bibliothèques dédiée à la recherche en vision par ordinateur. Elle est écrite et utilisée en langage C. Elle est constitué d'une multitude de programmes qui peuvent être chaînés pour produire une application — au minimum une source d'images, un programme de traitement, et un programme possédant une interface graphique sont ainsi liés. Chacun de ces programmes possède une API uniforme et constitue un composant.

Son objectif n'est pas directement la construction d'applications interactives, mais le support de la recherche en vision par ordinateur ; elle permet néanmoins de tirer quelques conclusions. Comme pour ImaLab, les auteurs veulent permettre le prototypage rapide d'algorithmes de vision par ordinateur ; dans ce but, la bibliothèque fournit :

- des outils pour implémenter rapidement les algorithmes, réduisant ainsi l'effort de développement ;
- un mécanisme pour composer plusieurs algorithmes ;
- des outils pour construire une interface graphique permettant de visualiser les résultats d'exécution, réduisant ainsi l'effort d'évaluation.

Cette bibliothèque ne s'est cependant pas répandue parmi ses utilisateurs potentiels. Au vu de la section précédente, nous attribuons cet échec aux causes suivantes :

(a) un mauvais ciblage. Le besoin central des dévelopeurs en vision par ordinateur est le prototypage rapide, pas la visualisation d'informations. L'écriture

d'algorithmes de traitement d'image est une tâche pénible ; le processus de développement est en général celui de l'essai et erreur. Par conséquent il n'est pas cohérent d'allonger encore le cycle développement-compilation-exécution avec des outils de construction d'interfaces graphiques et de visualisation.

(b) un seuil élevé. Utiliser Vista requiert pour Patrick d'apprendre, en plus du langage C et des extensions de la bibliothèque, un langage particulier qui représente la structure des entrées et des sorties d'un algorithme. En outre, si le support de la visualisation des résultats est très développé, peu d'outils sont fournis pour simplifier le développement des algorithmes de traitement.

Pour construire des prototypes de vision plus complexes, implémentant l'un des services qui nous interessent, il est nécessaire de manipuler un troisième langage d'assemblage : un langage de commande, ou *shell*, sous UNIX. Il permet de chaîner des algorithmes plus simples, c'est-à-dire les composants de la bibliothèque. Dans le contexte d'utilisation de la bibliothèque, cette approche permet d'obtenir un seuil faible pour Laurence. Un service est encapsulé dans un programme unique (un script), qui produit des événements d'une manière conventionnelle (sur sa sortie standard), sous forme de texte.

Une des conséquences négatives est que les choix technologiques effectués pour Vista confinent les utilisateurs dans l'environnement UNIX / X11, et sur une machine unique, ce qui n'est pas forcément le choix retenu par les clients.

Notons que Vista fournit des performances très mauvaises, notamment parce que les outils d'implémentation d'algorithme de traitement d'image ne sont pas performants, et parce que la communication entre composants de traitement d'image requiert la sérialisation et la dé-sérialisation des images intermédiaires.

Nous retenons néanmoins un point fort de cette bibliothèque : le souci de rendre *observable* le fonctionnement de l'algorithme de vision, sous la forme d'outils de déboguage (graphique en particulier). Ceci est une aide au prototypage rapide (pour Stanislas).

4.2.3.2 Architecture basée sur COM+

[Economopoulos et Martakos, 2001] proposent d'utiliser une infrastucture bien établie comme support de la conception et de l'implémentation de systèmes interactifs basés sur la vision. Il s'agit de l'architecture DNA™ de Microsoft (*Distributed Internet Application Architecture*).

Cette approche à composants répartis permet la délocalisation du service perceptif, qui devient indépendant de tout client applicatif. L'architecture d'un tel système est structurée en trois couches :

- le *presentation tier*, ou couche de présentation, où résident les clients applicatifs ou interfaces utilisateur ;
- le *business tier*, ou couche fonctionnelle, où résident les composants, ici implémentant un service perceptif ; un composant « racine » ou superviseur expose le service et ordonnance le fonctionnement des autres ;
- le *data tier*, ou couche données, est le lieu par lequel transite l'information entre composants.

L'objectif de cette approche structurelle est (implicitement) de baisser le seuil fonctionnel pour Patrick et Stanislas. En utilisant exclusivement des composants, la flexibilité et la modularité sont accrues, et le développement et le test de prototypes de services devient rapide. En outre, un service peut facilement être réparti sur plusieurs machines (si les besoins de calcul sont trop importants), éventuellement isolées de la machine « cliente », celle ou fonctionne le *presentation tier*.

Les auteurs admettent que « l'apprentissage des compétences de programmations nécessaires est difficile », c'est-à-dire que le seuil sera élevé, pour toutes nos catégories d'utilisateurs, et en particulier pour Laurence.

En outre, le gain de flexibilité est annulé par le modèle *3-tiers*. Si on le respecte de manière stricte, toutes les données (en particulier, les événements d'interaction et les images) doivent transiter par le SGBDR qui implémente le *data tier* (dans l'article, Microsoft SQL Server™). Ceci impose de multiples recodages, sérialisations, et copies des données, ce qui est irréaliste pour des traitements de vision par ordinateur à latence faible. Les auteurs choisissent de stocker les images successivement traitées sur le disque, et ne ne faire transiter dans la base de données que des références à ces fichiers : ils atteignent ainsi des performances acceptables, mais le modèle flexible esst brisé, et la distribution devient impossible ; en particulier la séparation des trois *tiers*.

Remarquons enfin que, si ce modèle n'est pas utilisable pour l'architecture interne de notre boîte à outils (c'est-à-dire pour Stanislas et Patrick), il l'est plus pour l'API externe (pour Laurence). En effet, les performances des architectures à composants réparties (évaluées dans le cas de CORBA dans [Vinoski, 1997]) permettent la construction d'applications même pour des requis de latence très contraignants, pour les implémentations les plus performantes des middlewares. Malgré tout le coût de l'apprentissage reste prohibitif.

4.2.3.3 Conclusions sur les approches à composants

À l'instar des deux exemples de cette section, les architecture et middlewares à composants existants sont bâtis sur des sémantiques point-à-point et synchrones.

Les APIs synchrones sont peu adaptées à Laurence, habituées aux modèles à événements et à l'asynchronisme des boîtes à outils de construction de GUIs. Ils la contraignent à utiliser un modèle de programmation multi-tâche, plus lourd, et élèvent donc le seuil d'utilisation. En outre, ce modèle est peu robuste à un éventuel échec du système de perception.

D'autre part, les architectures à composants imposent de sérialiser tous les messages échangés entre Patrick et Laurence. Ceci n'est pas réaliste en termes de performances dans les cas où le volume de l'information transférée est important. La communication point-à-point interdit évidemment la diffusion multiple (*multicasting*) efficace de messages, ce qui est problématique si un service perceptif est utilisé par plusieurs applications interactives.

Enfin, l'utilisation d'un architecture à composants moderne impose à Laurence un seuil trop élevé. Bien que des *bindings* pour de multiples langages de programmation soient fournis, il lui sera nécessaire, au minimum, d'apprendre et d'utiliser une API pour instancier un composant, voire un langage de description d'interface (IDL) ; ou encore d'intégrer son application à un environnement de développement spécifique (plate-forme .NET ou Eclipse par exemple). Symétriquement, Patrick devra fournir un effort d'intégration comparable pour fournir un composant.

4.2.4 Approches orientées service

L'organisation pour le progrès des standards de l'information structurée (OASIS : *Organization for the Advancement of Structured Information Standards* [OASIS, 2006]) définit les architectures orientées service (Software Oriented Architecture ou SOA) de la façon suivante :

> L'Architecture Orientée Service est un paradigme pour l'organisation et l'utilisation de capacités [logicielles] qui peuvent être sous le contrôle de différents propriétaires [à la conception ou au déploiement]. Elle fournit un moyen uniforme d'offrir, découvrir, interagir avec, et utiliser ces capacités, en produisant les effets désirés, de manière cohérente avec des préconditions et postconditions mesurables.

D'après le modèle de référence publié par OASIS [MacKenzie et al., 2006] deux des concepts essentiels qui permettent de décrire le paradigme SOA sont les suivants :

visibilité Ceux qui ont un besoin (Laurence) et ceux qui peuvent fournir des services (Patrick) doivent se voir et se comprendre. Ceci est généralement obtenu en fournissant des descriptions formelles des fonctions, requis techniques, contraintes, et mécanismes d'accès ou de réponse. La visibilité introduit donc la possibilité de mettre en correspondance les besoins et les services.

interaction L'interaction (avec un service) est l'activité qui consiste à utiliser une capacité logicielle, typiquement *via* l'échange de messages entre unités logicielles.

Le troisième concept (l'*effet* sur le monde réel) ne s'applique pas au cas des services perceptifs : par nature, leur rôle n'est pas d'agir. C'est à l'application interactive de d'agir en fournissant un retour d'information et une interface à l'utilisateur.

Pour nous, le terme de SOA désigne une approche à l'architecture logicielle modulaire qui définit l'utilisation de resources logicielles (ou *services*) faiblement couplés. Ces ressources sont accessibles (en général sur le réseau) de manière indépendante de leur implémentation. Une SOA n'est pas liée à une technologie spécifique et peut être implémentée en utilisant l'un des nombreux standards d'interopérabilité existants, voire de manière *ad hoc*.

4.2.4.1 Peripheral Display Toolkit

PTK [Matthews et al., 2004] est une bibliothèque qui permet la création d'applications interactives du type *affichage périphérique*. Elle intègre des fonctionnalités de perception de l'activité, de traitement, et d'action. C'est un prototype, résultat d'un projet de recherche académique. Elle possède une architecture mixte composants/services. Des services, au sens SOA, emballent les périphériques d'entrés : sur l'exemple de la figure 4.1, une caméra. Ils fournissent des données brutes aux applications. La bibliothèque fournit ensuite trois types de composants aux applications :

- des abstracteurs (*abstractors*) traitent les données issues des capteurs, en réduisant le volume d'information et en le rendant lisible par un client logiciel ou par un humain ;
- des notificateurs (*notification maps*) filtrent les données résultantes selon leur pertinence (dans PTK, selon l'importance de l'information mesurée relativement à l'attention de l'utilisateur) et génèrent des événements d'interaction ;
- des adaptateurs de sortie (*output wrappers*) produisent un effet (typiquement un retour visuel) en consommant ces événements.

D'après notre définition de service (chapitre précédent), l'abstracteur est un service perceptif : les éléments logiciels situés en aval (sur le flot de l'information perçue) constituent le client applicatif. Comme ARToolkit, PTK intègre tous les éléments d'une application perceptive, de l'observation à l'action ; ce n'est néanmoins pas un monolithe car c'est à Laurence d'intégrer les différents composants.

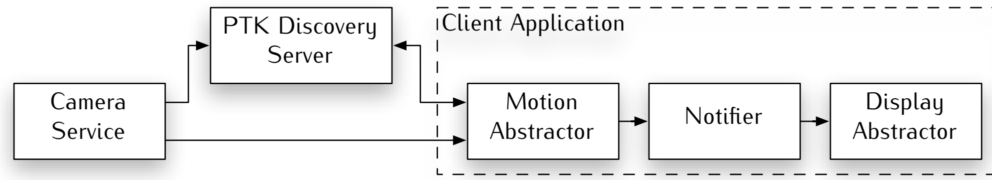


Figure 4.1 . Exemple de déploiement d'un système interactif utilisant PTK.

Le flux video issu d'une caméra est acquis pas un service de capture. Après découverte du service *via* un serveur de découverte, il est transmis à l'application cliente. Un composant *abstracteur* évalue quantitativement le mouvement dans la scène. Un composant *notificateur* interprète sémantiquement cette quantité de mouvement (aucun, peu, ou beaucoup de mouvement) et génère des requètes de notification à partir de cette interprétation (de *ignorer* à *interrompre l'utilisateur*). Ces requètes sont transmises à un dernier composant, qui affiche une transition correspondante sur le périphérique de sortie : dans l'exemple, une *Ambient Orb*.

L'abstraction est fondamentale pour Laurence dans PTK : elle fournir une information sémantiquement compatible avec les questions que le client peut se poser. Elle permet donc d'interprêter l'activité de l'utilisateur pour construire l'interaction. Le mécanisme d'accès au service d'entrée dans PTK permet de spécifier si l'abstraction doit avoir lieu en amont (i.e. dans le service capteur) ou en aval (dans l'application).

Il existe une contradiction dans cette approche : d'une part, l'auteur remarque qu'il peut être nécessaire de réutiliser les entrées, abstraites ou non, dans plusieurs applications. D'autre part PTK impose soit d'effectuer l'abstraction au niveau du capteur, auquel cas les données brutes (le flux video par exemple) ne sont plus accessibles ; soit d'effectuer l'abstraction dans l'application, auquel cas les données brutes doivent être distribuées à tous les clients, et le processus d'abstraction éventuellement répliqué dans plusieurs clients.

Dans notre cas, les données brutes sont toujours un flux d'images. Pour la plupart des services il est important de minimiser la latence de l'abstraction, et de maximiser la fréquence de traitement. Le processus d'abstraction doit donc être colocalisé avec celui d'acquisition.

Pour conclure sur PTK, on peut constater que l'architecture mixte composants/services permet d'atteindre un seuil très faible (tous les exemples présentés dans l'article requièrent moins de 40 lignes de code) et un haut niveau de réutilisabilité des composants. Par contre, il convient de bien placer la frontière entre architecture à services et architecture à composants.

4.2.4.2 EasyLiving et InConcert

L'environnement intelligent EasyLiving [Brumitt et al., 2000] est une architecture et un ensemble de technologies perceptives pour l'informatique ubiquitaire (figure 4.2). Son objectif est la création d'environnements intelligents. Grâce en particulier à des services d'identification visuelle et de suivi d'utilisateurs, il fournit à ses utilisateurs des services tels que « imprime sur l'imprimante la plus proche de moi », ou encore

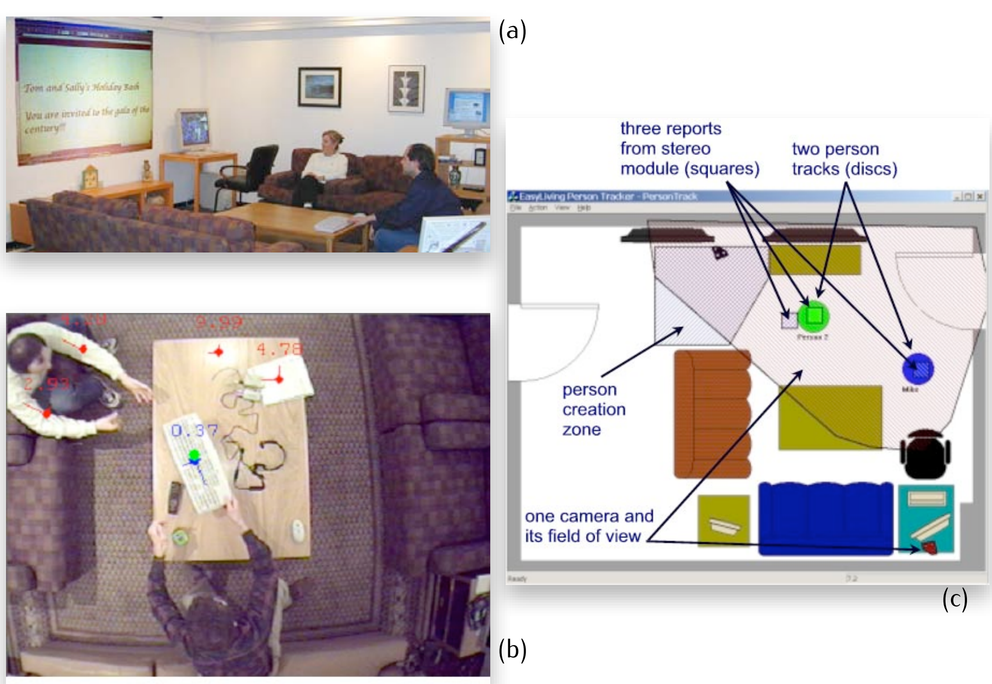


Figure 4.2 . L'environnement intelligent *Easy Living*.

D'après [Brumitt et al., 2000]

« affiche ma présentation sur le projecteur de cette salle de réunion ». Les auteurs utilisent les termes *service* et *composant* de manière interchangeable.

Il est bâti sur le middleware InConcert [Sarin, 1996], qui fournit :

- un mécanisme d'attribution de noms, qui permet un adressage des services indépendant des machines et du réseau ;
- un système de passage de messages asynchrone entre services, où la source et la destination du message utilisent le mécanisme de nommage ;
- un langage de messages défini par un schéma XML.

Dans EasyLiving, chaque dispositif (caméra, lumière contrôlée, vidéoprojecteur) est encapsulé dans un service qui permet de le contrôler. Des services perceptifs fonctionnent en permanence (sur des machines différentes) pour abstraire les données brutes issues des services-dispositifs. Les concepteurs choisissent de produire une sortie générique pour les services comparables : un service encapsulant un *trackball*, une souris, ou un système perceptif de détection de geste de pointage produisent le mêmes sorties.

Pour utiliser un service, il est nécessaire de le découvrir — c'est-à-dire se connecter au serveur annuaire, puis de trouver le service adapté. Pour ce faire, InConcert impose aux services de fournir une description fonctionnelle formelle, comparable à une description de son API : les commandes qu'il supporte, les paramètres acceptés, et les sorties qu'il produit. Cette description est écrite en suivant un schéma XML prédéfini.

Dans notre contexte, la possibilité de se connecter de manière simple au service perceptif est interessante : le seuil est bas, puisque tout environnement ou langage de programmation dispose des outils pour connecter une application à un service *via* TCP/IP. Le coût supplémentaire de l'interaction avec un service d'annuaire paraît compensée par le fait que le client est dispensé des contraintes et aléas de l'adressage d'un service sur TCP — en particulier, un service peut être migré sur une machine différente sans intervention du développeur client. Par contre, il paraît difficile de demander à Laurence de déployer un serveur annuaire en plus du service perceptif.

L'abstraction et la généralisation des sorties des services perceptifs (comparable aux objectifs de Papier-Mâché) permet également de baisser le seuil, et d'augmenter la flexibilité du système interactif. Les services perceptifs deviennent interchangeables, ce qui permet le prototypage rapide.

Il paraît cependant contraignant d'imposer à Laurence et Patrick de manipuler des descriptions de services et événements dans un langage dérivé de XML. Outre la charge de calcul accrue (chaque message-événement doit être décodé par Laurence), la charge de développement est plus lourde puisque les deux parties doivent manipuler une bibliothèque dédiée à la manipulation de XML.

4.2.4.3 Architecture à service pour robots mobiles

Dans [Melchior et Smart, 2004], les auteurs présentent une approche de conception et une architecture orientée service pour le logiciel embarqué sur un ou des robots mobiles autonomes. Nous la nommons MRt (pour *Mobile Robot Toolkit*) par souci de concision. Il paraît nécessaire de démonter la pertinence de ce projet dans notre état de l'art.

Un robot embarque un ou des capteurs (caméra, sonar, télémètre laser), des moyens de calculs, un logiciel qui le contrôle, et des actuateurs (moteurs). Sur le plan logiciel, il a donc des besoins de perception, dont certains sont basées sur la vision. Les auteurs proposent de construire une boîte à outils verticale pour la construction (logicielle) d'un robot. Comme nous, ils s'adressent à des classes d'utilisateurs différentes. On peut faire le parallèle entre Caroline et le robot lui-même ; entre Laurence et le roboticien qui conçoit le programme « maître » du robot ; et Patrick et Stanislas jouent les mêmes rôles que dans notre cas.

Certains services sont comparables ou identiques (détection d'objets, localisation, etc.) Les besoins non fonctionnels le sont également, tant pour les services que pour l'architecture de la boîte à outils. En particulier, la latence doit être faible (pour que le robot puisse être réactif), et les services comme l'architecture doivent être autonomes (puisque l'objectif est la construction d'un robot autonome).

MRt propose à Laurence de manipuler exclusivement une API basée sur l'abonnement à des événements issus de services. Une classe abstraite de motif adaptateur, nommée `Client`, permet de communiquer avec un service. En substance, à la création d'un `Client`, Laurence désigne de manière informelle le service qui l'interesse pour s'y connecter, puis reçoit les événements de perception sous forme d'appel à des méthodes de `Client`. Afin de réduire plus avant le seuil de MRt, les auteurs fournissent des implémentations de l'adaptateur pour de nombreux langages de programmation. Le travail de Patrick est symétrique : il doit implémenter une classe concrète dérivée de `Interface`, l'interface abstraite générique d'un service. Pour résumer, le couple `Client/Interface` constitue un mécanisme asynchrone d'appel distant de méthodes.

En ligne, c'est un programme superviseur, le MCP (*Master Control Program*) qui est chargé de

– la découverte de services ;
– l'adressage des services (traduction entre noms et adresses des services) ;
– l'établissement de la connection entre services et clients ;

parmi d'autres tâches qui ne nous concernent pas directement (diagnostic vital des services, auto-réparation, suivi des performances, etc.) Son rôle est donc comparable à un ORB CORBA.

L'ensemble formé par le MCP et les implémentations des adaptateurs (`Client` et `Interface`) forme un middleware orienté service.

La communication entre un service et son client est assurée par un mécanisme d'IPC (*Interprocess Communication*) conçu par les auteurs pour minimiser la latence. Les événements sont encapsulées dans un message XML. Son fonctionnement est volontairement simpliste (la communication a lieu exclusivement sur TCP, et chaque message est simplement préfixé par un en-tête textuel en ASCII), afin de préserver la possibilité d'interopérer avec d'autres programmes n'utilisant pas le middleware.

L'approche proposée à pour conséquence un seuil bas pour Laurence comme pour Patrick : elle est agnostique en termes de langage de programmation et d'environnement de développement. Elle fournit aux différentes catégories de développeurs un chemin de faible résistance (la manipulation des adaptateurs fournis). Elle préserve la possibilité d'atteindre un plafond fonctionnel élevé *via* un chemin de plus forte résistance, en implémentant des adaptateurs différents qui utilisent le mécanisme d'IPC simplifié. Elle permet enfin d'adopter une approche mixte composants/services en utilisant la même API pour les composants et les services. Par contre, elle impose à Laurence de manipuler un logiciel dédié au fonctionnement du middleware (le MCP), ce qui alourdit le déploiement ; mais ce MCP permet également de mesurer et de réguler la qualité de service fournie par les différentes composantes du système.

4.2.4.4 Conclusions sur les approches orientées service

Le paradigme SOA est fonctionnelement similaire aux approches à composants. La différence essentielle est la granularité du système construit. Les atomes d'un système bâti sur le modèle SOA sont plus « gros » ; c'est-à-dire qu'ils sont d'un niveau d'abstraction plus élevé, donc moins fortement couplés. SOA introduit des notions comparables aux requis centrés utilisateurs : des mesures de qualité de service (QoS), et le contrat passé entre le service et l'application cliente (en termes fonctionnels et de QoS). Ceci permet, pour reprendre les critères de Myers, à Patrick de « viser juste » en exprimant une description de son service, et de baisser le seuil fonctionnel pour Laurence.

Bien que le paradigme SOA soit agnostique en termes de plate-forme logicielle sous-jacente, c'est-à-dire du middleware supportant la découverte et l'interopération entre services, les implémentations existantes utilisent ou reproduisent des middlewares à composants répartis. Elles sont basées par exemple sur CORBA ou sur DCOM. En fournissant à Patrick et Laurence des adaptateurs permettant la construction ou la connection à un service, il est possible de baisser le seuil et/ou de fournir des chemins de résistance faible pour le développement. Par contre, le déploiement d'un système SOA impose généralement de déployer un service spécifique au middleware (typiquement un service d'annuaire, et/ou un superviseur).

4.3 Synthèse : Requis sur l'architecture

Avant de faire le choix d'une approche structurelle et d'une implémentation, nous nous appuyons sur notre état de l'art pour résumer les requis que doit satisfaire l'architecture de notre boîte à outils *gmlVision*. Nous présentons tout d'abord des requis non fonctionnels (latence, autonomie, et fiabilité), qui découlent des besoins des services. L'étude des boîtes à outils existantes nous permet ensuite de proposer quatre requis fonctionnels pour une architecture adaptée : asynchronisme, abstraction, isolation, et contrat. Enfin, nous résumons les forces et faiblesses des quatre approches présentées ci-avant vis-à-vis de ces requis.

Nous avons brièvement présenté ces requis [Borkowski et Letessier, 2006] lors d'une étude préalable sur la conception centrée utilisateur d'un système interactif basé sur la vision par ordinateur.

4.3.1 Requis non fonctionnels

Les requis des services contenus dans une bibliothèque imposent des contraintes sur l'architecture de cette bibliothèque. Plus précisément, pour que ces services puissent atteindre la QoS requise, il est nécessaire que l'architecture logicielle intégrant la bibliothèque et son client (l'application interactive) l'autorise.

4.3.1.1 Latence

Les systèmes interactifs rencontrés imposent tous une borne supérieure de latence admise. Pour les plus contraingnants, cette borne est de 50 ms ; pour les moins contraingnants, de 1 s environ. Lorsque le système respecte ces bornes de latence, l'utilisabilité peut être optimale. Rappelons que la latence dont nous parlons est la latence **totale** du système interactif, perception comprise (c.f. figure 3.17 page 56). Elle inclut donc les temps :

- d'acquisition (matérielle) du signal video ;
- de traitement (logiciel) par le service perceptif ;
- de communication avec le client applicatif ;
- de traitement par le client ;
- de production d'un retour (*feedback* visuel par exemple).

D'après [Borkowski, 2006] et les mesures de [Borkowski et al., 2004], la latence totale d'un tel système sans aucun temps de traitement est de 32 ms $\pm$ 25 ms (caméra Sony EVID-100P, connectée à un videoprojecteur Epson sans ordinateur intermédiaire). Cette latence est due à parts environ égales à l'intégration du signal par la caméra et au temps d'affichage. L'écart-type observé est du à la désynchronisation entre l'acquisiton (ici, à 30 Hz) et l'affichage (ici, 60 Hz). Les opérations de transfert et d'aquisition par l'ordinateur, de traitement de l'image par le service perceptif, de communication par l'architecture ou le middleware, et de traitement des événements par le client applicatif, doivent donc se partager, en moyenne, les 18 ms restantes pour rester en-deça du seuil des 50 ms.

Les opérations de traitement seront en général coûteuses en temps de calcul. Le traitement permettant d'extraire la silhouette d'une personne d'une image PAL, utilisé pour implémenter *VideoPlace* [Krueger et al., 1985] sur une machine moderne (Intel Xeon™ 2.8 GHz), requiert 9 ms de traitement en moyenne. Un traitement avant l'affichage, même trivial, requiert plusieurs millisecondes. Par conséquent, la surcharge due à l'architecture (*overhead*), et en particulier le temps de communication avec le client applicatif doit être le plus réduit possible. Dans le cadre des expériences décrites au chapitre 7 page 143, nous avons constaté qu'une latence de communication de 2 à 3 ms était acceptable.

Ceci peut être obtenu avec n'importe quelle approche, pourvu que le flot d'information transmis soit de volume faible. Dans le cas où un flux d'images doit être transmis (toujours dans l'exemple de *VideoPlace*), les approches qui imposent la sérialisation ou l'encodage des données échangées ne sont pas acceptables. Par contre, le middleware utilisé devra être « léger », c'est-à-dire ne pas ou peu transformer l'information qui circule entre le service perceptif et l'application, et entre composantes du service perceptif. Les approches les plus performante en termes de latence sont les approches classiques et monolithiques.

4.3.1.2 Autonomie

Les systèmes de vision par ordinateur nécesitent en général l'intervention d'un opérateur human pour la mise en place initiale et la maintenance en fonctionnement. Cette situation peut être acceptable dans le cadre du laboratoire expérimental (de vision par ordinateur), mais elle est très mal adaptée au déploiement de la vision — que ce soit dans le monde « réel » de l'industrie et du grand public, ou dans celui, théoriquement plus tolérant, de la recherche dans d'autres domaines (interaction Homme-machine, ergonomie, ou visualisation d'information par exemple).

Idéalement, Stanislas et Patrick (les fournisseurs de services de vision) devraient concevoir des solutions d'initialisation et de maintenance automatiques — au sens de la définition de l'autonomie donnée page 56. En particulier, une API pour la maintenance ne devrait pas être nécessaire.

Cependant, notre expérience en vision par ordinateur nous a appri que la conception de systèmes de vision automatiques (sans apprentissage encadré, sans calibrage manuel, sans reprise sur pannes assistée) est un objectif difficile. C'est d'ailleurs un défi de la recherche actuelle en vision qui ne sera pas atteint à court terme pour la plupart des problèmes.

Les services de perception visuel seront donc, en général, *coopérants* voire *assistés* (au moins à l'initialisation). Pour que le système reste utilisable, deux facteurs sont à prendre en compte lors de la conception :

- Caroline ou Laurence devront coopérer avec ou assister le système perceptif, mais elles ne sont pas des experts en vision. La collaboration devra donc être simple et minimale.
- Caroline mène a bien une tâche d'interaction. Si la coopération ou l'assistance est nécessaire, elle ne devra pas interrompre l'exécution de la tâche de manière significative.

Ceci impose quelques contraintes sur l'architecture. Les choix structuraux effectués ne doivent pas diminuer l'autonomie du système ; en particulier les mécanismes de reprise sur panne devront être invisibles pour le client applicatifs. Elle pourra éventuellement améliorer l'autonomie du système global en préservant (rendant persistants) les réglages effectués lors de l'assistance à l'initialisation. D'autre part, l'architecture devra permettre la coopération et la collaboration avec le système perceptif, en fournissant des mécanismes adaptés.

4.3.1.3 Fiabilité

Dans la plupart des cas, l'information doit être transmise de manière fiable depuis le système perceptif jusqu'à l'application interactive. Par exemple, une application qui reçoit des événements lorsque une personne entre ou sort d'une pièce augmentée ne peut se permettre de « manquer » un événement : son état interne deviendrait alors incohérent. D'un autre côté, la perte occasionnelle de quelques événements (par exemple : événements de suivi pour une application fournissant une interaction fortement couplée) peut détériorer l'expérience utilisateur sans pour autant briser l'interaction ou interrompre la tâche de l'utilisateur.

Par conséquent, l'architecture concrète retenue doit fournir un mode de transport fiable de l'information ; un compromis entre fiabilité et latence est acceptable dans certains cas.

D'autre part, des échecs de perception sont toujours possible, en particulier dans le cas de la vision. La caméra peut être déplacée ou occultée inopinément, par exemple. Il doit donc être possible de notifier à tout moment l'application cliente de tels échecs, et ce de manière fiable.

Enfin, dans le cas où le service de perception et l'application cliente sont délocalisés, il est possible que la connection entre les deux soit rompue. L'architecture doit donc là encore être capable de notifier services et applications de l'état des connections, ainsi que de fournir un mécanisme de (re-)connection automatique.

4.3.2 Requis fonctionnels de l'architecture

Les analyses centrées utilisateur présentés par certains auteurs de notre état de l'art, l'analyse de leurs boîtes à outils, et leur succès éventuel, nous amènent à proposer quatre requis qui synthétisent les besoins de Laurence, et dans une moindre mesure de Patrick. Ces requis guiderons notre choix d'une architecture logicielle, puis les choix techniques dans la conception de *gmlVision*.

4.3.2.1 Asynchronisme et communication par événements

Nous nommons *événement* une pièce d'information atomique, étiquetée par un point dans le temps, généré par un service (perceptif ou contextuel) ou par une application, qui peut être distribué à un ou plusieurs clients (autres services ou applications). Typiquement un événement possède un nom (la classe dont il est une instance) et un ensemble de données (ses caractéristiques propres en tant qu'instance, en particulier son étiquette temporelle ou *timestamp*).

Par exemple, dans la boîte à outil graphique *X window* [Scheifler et Gettys, 1986], le survol d'un widget par le pointeur de la souris déclenche un événement de classe `Enter`. Cet événement est étiqueté, en particulier, par l'identifiant du widget concerné et les coordonnées du pointeur de la souris. Ici la source de l'événement est le service perceptif, inclus dans le serveur X, qui abstrait le pilote de la souris.

Ce paradigme, rendu populaire par *X window* et la *Macintosh Toolbox* [Apple Computer, 1992] est universel dans les boîtes à outils classiques de construction d'interface graphique. Son fonctionnement est représenté sur la figure 4.3. Il paraît naturel de l'utiliser pour l'interface de notre boîte à outils : Laurence est en effet familière avec ce paradigme. L'utiliser évite l'apprentissage d'un autre paradigme.

D'autre part, la perception (artificielle) ainsi que les besoins de notification évoqués plus haut sont naturellement asynchrones. Un événement peut avoir lieu dans l'environnement observé à tout instant. De même, un service (ou le middleware utilisé) peut devoir informer l'application cliente d'un changement d'état à un instant non prévisible.

Il est donc requis d'utiliser un système à événements pour la conception de notre boîte à outils. Remarquons que le mode de transport choisi des événements devra

permettre de véhiculer des méta-informations de fort volume (de type flux video en particulier).

4.3.2.2 Abstraction

Pour les dévelopeurs en interaction Homme-machine, « l'abstraction des entrées est la partie la plus difficile et coûteuse en temps du développement d'une application » [Klemmer et al., 2004]. Comme nous supposons que Laurence, utilisateur principal de la boîte à outils (le développeur de l'application interactive), n'a aucune expertise en vision par ordinateur, les informations spécifiques à la vision ne devraient pas être visibles dans l'API. Leur manipulation ne doit en aucun cas être nécessaire.

Même si les résultats de perception sont rendus plus riches par l'adjonction, par exemple, d'un indice de confiance, cette information n'est pas pertinente pour Laurence. Elle ne dispose pas de la connaissance, ni des métriques pour la traiter. Par exemple, les souris optiques utilisent l'algorithme ZNCC (corrélation croisée normalisée) pour déterminer la direction et l'amplitude d'un mouvement à chaque pas de temps. Le coefficient de corrélation est un résultat naturel de cet algorithme, mais il n'est pas fourni au client (le pilote de la souris). Tant que ce coefficient est supérieur à un seuil, la souris fournit des événements de position au pilote ; sinon, elle reste silencieuse. Les données internes d'un service, en particulier les informations liées à la qualité de service, ne doivent pas être considérées comme des sorties essentielles, et par conséquent ne doivent pas être visibles par défaut. Ce raisonnement se généralise pour Patrick : lui peut avoir besoin d'accéder à ce type d'information (par exemple d'un indice de confiance dont il connaît la métrique), et le demander explicitement à un service déployé. Par contre il ne doit pas avoir conscience du choix d'un algorithme particulier ou de ses réglages.

D'autre part, les actions de l'utilisateur final (Caroline) capturées par un service perceptif doivent être abstraites sous forme d'événements interactifs qui soient de la même forme pour des techniques d'interaction similaires. Par exemple, un service de suivi de doigt (implémentant **S10**, c.f. page 46) doit produire des événements de mouvement compatibles avec ceux d'une souris, ou d'un suivi de jetons (**S4'**, c.f. page 48). Les données abstraites générées par un service perceptif doivent être exprimées dans un formalisme aisément manipulable par d'autres boîtes à outils, et connectable a des applications interactives existantes, afin de

- permettre le prototypage des applications interactives sans le système perceptif (i.e. en utilisant des périphériques classiques) ;
- permettre la substitution d'un service perceptif par un autres (par exemple pour explorer plusieurs modélités ou techniques d'interaction).

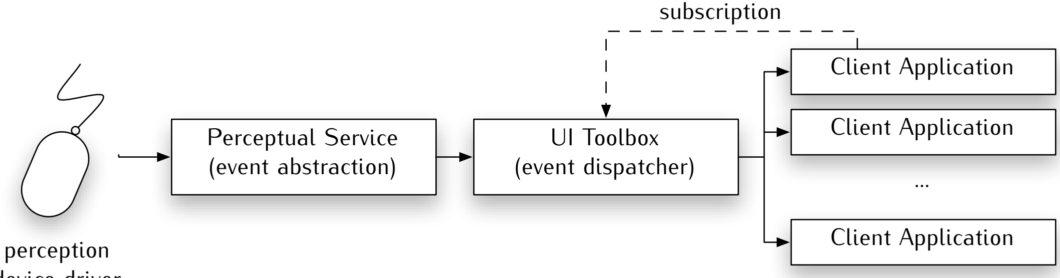


Figure 4.3 . Paradigme abonnement / distribution d'événements.

Dans les boîtes à outils classiques, les pilotes de matériel perceptifs génèrent des événements qui sont abstrait par une couche perceptive. Ce service est souvent réduit ou inexistant grâce à l'existence de la norme *USB HID* pour les périphériques de pointage, gérée par les pilotes. La boîte à outils dispose d'un « distributeur d'événements » auprès duquel s'abonnent les clients ; chaque événement est alors répliqué et distribué à tous les abonnés.

En conclusion, l'API d'un service perceptif doit

a. fournir une information sémantiquement compatible avec les questions que le client peut se poser [Matthews et al., 2004]
b. rendre invisible le fait que la perception est basée sur la vision par ordinateur ; et
c. généraliser la formulation des événements pour une tâche de perception donnée.

4.3.2.3 Isolation

Un service perceptif peut devoir être utilisé par de multiples applications interactives, éventuellement fonctionnant sur de multiples machines — au même titre que la souris ou le clavier dans des applications WIMP. Par exemple, dans le cas de S16 (page 53), le suivi de la tête de l'utilisateur peut être utilisé pour contrôler le défilement dans un navigateur web, et pour centrer la vue sur le visage de l'utilisateur dans une application de videoconférence *VC*. En imaginant qu'un service de capture video sous-jacent est utilisé, il sera également utilisé par S16 ainsi que directement par *VC*. Cet exemple est représenté sur la figure 4.4.

[Klemmer et al., 2004] résume cette situation ainsi : « une information [perceptive] d'entrée particulière peut être utilisée pour de nombreuses différentes sorties. » Ce constat impose que l'information générée par un service perceptif doit pouvoir être partagée, en restant accessible localement et à distance (i.e. sur une machine et/ou dans un lieu différents). En outre, la réutilisation ne se fait pas nécessairement entre clients finaux, c'est-à-dire entre applications interactives : les clients peuvent être d'autres services perceptifs. Un service doit donc facilement pouvoir être étendu, par exemple au travers des patrons de conception *adaptateur*, *proxy*, *aggrégateur*, ou *superviseur* ; en particulier être inclus dans un autre service ou une application.

4.3.2.4 Contrat

Les concepteurs d'une boîte à outils ou d'un périphérique d'entrée établissent un contrat avec Caroline et Laurence. Ce contrat est constitué de critères non fonctionnels qui décrivent les caractéristiques d'un service. Pour un dispositif physique, par exemple pour une souris optique, il s'agit :

– pour Caroline, des conditions d'utilisation, c'est-à-dire des caractéristiques de l'environnement qui sont contraignantes pour le dispositif : type de surface sur lesquels l'utiliser, température, hygrométrie, etc ;
– pour Laurence, des caractéristiques de sa sortie pertinentes pour les techniques d'interaction mises en oeuvre : résolution de la souris, latence et fréquence de l'émission des événements positionnels, précision, stabilité statique, et autonomie.

Ce contrat peut aussi être implicite, en particulier pour un dispositif physique : l'utilisateur du service suppose alors que celui-ci fonctionne dans des limites acceptables d'utilisabilité, dans toutes les conditions réalistes.

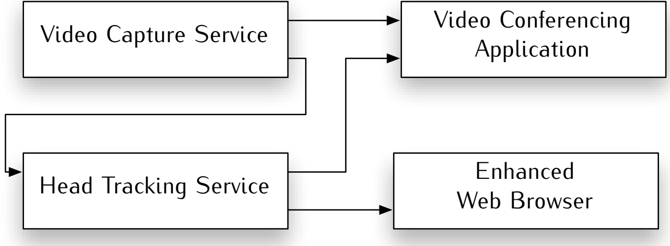


Figure 4.4 . Exemple de déploiement imposant l'isolation des services.

Deux services perceptifs (acquisition video et suivi de tête) sont utilisés de manière concurrente, respectivement par un autre service et une application interactive, et par deux applications.

Dans le cas des systèmes perceptifs basés sur la vision par ordinateur, les requis d'utilisabilité (en particulier la latence et/ou la stabilité) sont difficiles à remplir, et mal perçus par les fournisseurs de services. Il est donc nécessaire que Patrick spécifie, les limites dans lesquels le service qu'il conçoit fonctionne, c'est-à-dire les performances (ou caractéristiques non fonctionnelles) qu'il atteint dans l'environnement auquel il est destiné. Ceci correspond à une démarche d'évaluation binaire de qualité de service.

Il est possible que les services soient contraints à l'introspection pour remplir le contrat. Par exemple, un service perceptif doit notifier ses clients des échecs de perception liés à des défauts de robustesse, plutôt que de fournir une information incorrete ou déformée éiquetée d'un indice de confiance faible.

Remarquons que le contrat peut être exprimé en utilisant la taxonomie introduite au chapitre précédent. Le contrat est donc une *spécification* du service peceptif. Comme elle n'est utilisée, dans notre cadre, que par Laurence, et au moment de la conception seulement, il n'est pas indispensable que le contrat soit numérique, ni formel ; aucun besoin de traitement automatique de la spécification n'existe pour construire les systèmes interactifs que nous avons rencontré.

4.3.3 Choix de l'approche

Nous pouvons à présent mettre en regard les différentes approches structurelles présentées et les requis que nous avons proposé ci-dessus.

asynchronisme et événements Les quatres familles d'approches présentées permettent l'utilisation d'un formalisme asynchrone à événements. Dans le cas des monolithes, c'est d'ailleurs le seul formalisme utilisé : Laurence écrira simplement des procédures de rappel (*callbacks*) pour gérer les événements entrants. Dans les autres cas, La bibliothèque devra fourni des adaptateurs (équivalents à l'approche monolithique) pour permetter à Patrick d'émettre les événements et à Laurence de les recevoir. Ces adaptateurs sont les éléments logiciels fondamentaux des architectures à composants et à services. Par contre, les middlewares existants sont presque tous synchrones.

abstraction Aucune des architectures présentées ne permet *per se* de satisfaire le requis d'abstraction. Il est imaginable de définir des formulations abstraites de chacune des familles d'événements de perception émises par les services de la boîtes à outils, et d'imposer à Patrick et Laurence de ne manipuler que ces formulations. On crée ainsi un chemin de moindre résistance pour Patrick comme pour Laurence. Ceci est possible dans le cas des architectures à composants et à services, en ne permettant la génération et la circulation que d'événements d'un format donné.

isolation Les bibliothèques classiques et monolothiques ne permettent pas de concevoir des services isolés. Les implémentations existantes d'architectures à composants ou à services permettent de concevoir les éléments perceptifs d'un système interactif comme des « boîtes noires », dont l'état interne et le fonctionnement est invisible et indépendant de l'application interactive — ce qui renforce en outre le requis d'abstraction.

contrat Certaines architectures concrètes à composants ou à services intègrent une notion de contrat, sous forme de définition formelle de la fonction du service, et éventuellement de la qualité de service (en termes de sécurité, fiabilité, ou performance). Comme nous l'avons exposé plus haut, dans notre cas un contrat formel n'est pas nécessaire. Il est suffisant que l'API fasse référence au contrat informel à la connection à un service. Par exemple, Laurence pourrait spécifier qu'elle veut utiliser une implémentation de **S14**.

4.4 Conclusion

Dans ce chapitre, nous avons étudié différentes boîtes à outils de services de vision pour l'interaction et leurs architectures vis-à-vis des critères informels proposés par Myers. Nous avons dégagé des requis non fonctionnels (latence, autonomie, fiabilité) et fonctionnels (asynchronisme, abstraction, isolation, contrat) pour l'architecture et la conception d'une boîte à outils de services adaptée aux besoins de Laurence de de Patrick.

Il semble que les approches à composants et les approches à services soient à même de satisfaire nos besoins. La frontière les deux approches est ténue, et la différence tient essentiellement dans la granularité des services fournis, et dans le moment où l'on les exploite : typiquement, les architectures à composants favorisent la réutilisation logicielle au moment de la conception, et les architectures à services favorisent la réutilisation et la distribution au déploiement des applications.

En conclusion, il nous paraît adapté d'adopter une architecture orientée service pour fournir les services perceptifs à Laurence, et une architecture à composants pour l'API interne fournie à Patrick pour bâtir les services.

Afin de rendre la bibliothèque utilisable, il conviendra de fournir à Laurence des adaptateurs logiciels permettant la connection à un service de manière triviale, et de permettre également une interopération simple avec les services sans ce middleware.

5 Conception de la boîte à outils *gmlVision*

> *« Over time, I've come to the conclusion that the most important general interface design guideline is this : Make interfaces easy to use correctly and hard to use incorrectly. (. . .) Responsibility for interface usage errors belongs to the interface designer, not the interface user. »*
>
> *[Meyers, 2004]*

Nous avons défini les requis fonctionnels et structurels qu'une boîte à outils de vision artificielle pour l'interaction doit satisfaire. Dans ce chapitre nous présentons la conception de notre boîte à outils nommée *gmlVision*. Notre objectif est de parvenir à la spécification des fonctions offertes par notre boîte à outils et la spécification de son interface de programmation (API).

Nous prenons ici le rôle de l'architecte logiciel, qui travaille au niveau d'abstraction au-dessus de celui du concepteur, pour répondre à la question : quelle interface fournir pour satisfaire les besoins de tous les utilisateurs ? « L'architecte logiciel a besoin [de décrire] de multiples vues du logiciel, pour les divers utilisateurs et utilisations » [Perry et Wolf, 1992]. En d'autres termes, nous devons proposer des méthodes, des interfaces, et des outils, pour nos trois classes d'utilisateur développeur (Laurence, Patrick et Stanislas).

La difficulté du design de la boîte à outils réside dans le **choix** des démarches de conception, technologies d'interface, et (pour l'implémentation) outils à utiliser pour satisfaire les requis sans compromis.

Nous ne prétendons pas ici apporter d'innovation radicale dans ces domaines. L'architecture que nous retenons (orientée services dirigées par les événements) n'est pas nouvelle même si jusqu'à présent elle était confinée au domaine des services web. Nous choisissons également de mettre en oeuvre des composants logiciels mais notre utilisation en est triviale. Notre contribution consiste à démontrer que ces choix sont applicables, et pertinents, pour la construction d'une boîte à outils de vision pour l'interaction.

Dans la section approche|ref :sec-conception-iter|, nous détaillons les choix de conception fondamentaux et la démarche qui nous permet d'atteindre un recouvrement fonctionnel de l'ensemble des besoins exprimés.

Ensuite, dans la section 5.2, nous justifions l'adoption pour *gmlVision* d'une interface externe orientée services, et d'une interface interne à composants légers. Ces interfaces s'adressent respectivement à Laurence, et Stanislas et Patrick.

Enfin, dans la dernière section (5.3), nous présentons un ensemble de services satisfaisant les besoins exprimés lors des deux chapitres précédents. Cet ensemble est obtenu par l'application la méthode présentée dans la première section du chapitre.

5.1 Notre approche

L'objectif de cette section est de déterminer, d'une part, quelle doit être la forme générale de l'interface de notre boîte à outils, et d'autre part quelle démarche employer pour que cette interface corresponde à un recouvrement fonctionnel des besoins exprimés.

Nous débutons cette section en présentant les catégories d'interfaces développeur (API) qui permettent de recouvrir fonctionnellement un besoin par une boîte à outils : les API génériques, et les API spécifiques ou *ad hoc*.

Nous montrons qu'il ne nous est pas possible de suivre l'approche générique classique, en particulier parce que la mise en oeuvre de la boîte à outils serait alors hors de portée de Laurence. D'autre part, nous expliquons en quoi l'approche *ad hoc* est pertinente par rapport à notre besoin.

D'autre part, nous montrons que l'approche *ad hoc* est suffisante, et plus satisfaisante pour le développeur client (Laurence). Enfin, nous présentons les problèmes liés aux choix des services permettant d'atteindre le recouvrement fonctionnel, et proposons d'adopter une démarche empirique et itérative.

Enfin, nous justifions et décrivons l'adoption d'une démarche itérative et empirique pour construire l'API spécifique qui permet de recouvrir « au mieux » notre besoin.

5.1.1 Interface générique vs. interface *ad hoc*

Lorsque Laurence est confrontée à *gmlVision*, son activité est la séquence des trois actions suivantes : premièrement, elle exprime un besoin de perception dans le formalisme de la taxonomie proposée au chapitre 3 (page 58). Elle instancie ensuite un service correspondant à ce besoin. Elle traite enfin le flux d'événements issu de ce service pour construire des techniques d'interaction. Nous devons choisir une manière de présenter à Laurence les services présents dans *gmlVision* ; deux catégories d'interface sont possibles.

La première est l'approche **générique**. Dans notre cas, elle consiste à fournir à Laurence un ensemble de « briques de base » de vision par ordinateur, qu'elle peut assembler pour construire un service. Ces briques sont atomiques, c'est-à-dire qu'il n'est pas pertinent de les découper en constituants plus simples, et elles ne sont pas redondantes. L'hypothèse non formulée lorsqu'on construit une boîte à outils de cette manière est que pour tout besoin exprimé par Laurence, il existe un ensemble de briques que l'on peut assemble pour produire le service approprié.

La seconde est l'approche **spécifique** ou *ad hoc*. L'idée est de fournir un ensemble de services prédéfinis dans la boîte à outils, chacun pour un volume précis de la taxonomie. Ceci peut introduire des redondances : l'intersection des volumes de la taxonomie auxquels correspondent deux services prédéfinis n'est pas nécessairement vide. L'hypothèse est alors que pour tout besoin exprimé par Laurence, il existe au moins un service fourni correspondant.

Vis-à-vis des critères de Myers, on peut affirmer qu'une API générique fournit un seuil et un plafond plus élevé qu'une API spécifique : Laurence doit apprendre et manipuler une API de plus faible niveau d'abstraction, mais peut réaliser des services potentiellement non fournis par une API spécifique.

5.1.1.1 Un parallèle avec BLAS et LAPACK

Afin d'éclairer le lecteur sur ce que nous entendons par « approche générique » et « approche spécifique », nous proposons ici un parallèle avec BLAS et LAPACK, deux boîtes à outils pour le calcul linéaire que nous considérons respectivement comme générique (BLAS) et spécifique (LAPACK).

Les BLAS [Lawson et al., 1979] sont des procédures qui calculent les opération élémentaires du calcul linéaire, telles que les multiplications entre vecteurs et matrices.

Elles sont largement utilisées, par exemple pour bâtir des bibliothèques qui résolvent des problèmes de plus haut niveau. BLAS est uniquement une API, implémentée de manière optimisée par les constructeurs de supercalculateurs, de microordinateurs, ou des équipes de recherche [Whaley et al., 2001]. Nous pouvons faire le parallèle entre BLAS et la couche de bas niveau de notre future boîte à outils, i.e. les primitives de traitement d'image et de vision par ordinateur. Stanislas en est le développeur et Patrick l'utilisateur principal. À notre sens, les BLAS constituent une boîte à outils générique : c'est à un utilisateur-développeur d'assembler les « briques » de calcul, sans utilité directe, pour produire un service utile.

LAPACK [Angerson et al., 1990] est une boîte à outils logicielle de résolution de problèmes numériques. Elle fournit des procédures pour résoudre des systèmes d'équations, calculer les valeurs et vecteurs propres, ainsi que différentes transformations et décompositions matricielles. Remarquons que LAPACK exploite de manière intensive BLAS. Cette fois, l'interface de LAPACK est l'équivalent de l'interface externe de *gmlVision* ; Patrick en est le développeur et Laurence l'utilisateur principal.

LAPACK adopte une approche spécifique. Par exemple, pour résoudre le problème des moindres carrés linéaire (qui consiste à trouver le vecteur x minimisant la norme euclidienne de $Ax - b$, pour A et b donnés), LAPACK propose plusieurs procédures distinctes. Il s'agit des procédures de la famille `GELS` (*GEneralized Least Squares*). Elles utilisent différentes méthodes de calcul et différentes structures de données, et diffèrent par leur temps d'exécution, leur coût en mémoire requise, et la précision du résultat. Ces critères sont comparables à la notion de qualité de service évoquée dans notre contexte. La tâche de l'utilisateur de LAPACK est comparable à celle de Laurence : étant donné un besoin en termes de fonction et de qualité, choisir un service. LAPACK est donc une boîte à outils spécifique.

5.1.1.2 Irréalisme de l'approche générique

Dans le cas de la vision par ordinateur, une brique n'est généralement pas réutilisable. C'est-à-dire qu'un composant logiciel utilisé pour construire un service de perception particulier ne pourra pas être utilisé tel quel pour un autre service.

Par exemple, la brique de détection de visage utilisée pour un service de suivi de personnes ne peut pas être utilisée pour détecter les jetons de la *Table Magique*, et réciproquement. En effet, les algorithmes à employer sont différents (ils sont fondés sur un modèle d'aspect pour le premier, sur un modèle de couleur et de forme pour le second), et applicables à des conditions environnementales différentes.

En pratique, dans l'existant, on observe que de nouvelles briques (et de nouveaux algorithmes de vision) sont mis au point pour chaque nouveau problème de perception. Par exemple dans OpenCV, on trouve trois solutions différentes pour détecter des lignes dans une image, avec des performances différentes (latence, précision) : il n'existe pas de brique générique de détection de lignes. Pour choisir puis utiliser l'un ou l'autre composant, il faut en outre être expert en vision par ordinateur. Assembler des briques de vision pour produire un service relève donc d'un domaine où Laurence n'est pas compétente.

5.1.1.3 Pertinence de l'approche spécifique pour nos besoins

Une approche spécifique est acceptable si un nombre limité de services suffit à satisfaire l'ensemble des besoins présentés au chapitre 3. Nous montrons plus loin que c'est le cas, en proposant un recouvrement fonctionnel (section 5.3). Cette approche est donc réaliste : il faut en plus qu'elle ne nuise pas à l'utilisabilité de la boîte à outils.

Pour Laurence, cette approche permet d'atteindre un seuil fonctionnel plus bas que l'approche générique. Déterminer un service à instancier revient à le choisir parmi un catalogue de services fournis plutôt que d'exprimer le besoin formellement. Le seuil reste bas tant que ce choix est simple, c'est-à-dire que Laurence

– peut identifier le service approprié d'après la description de son besoin ;
– n'est pas confronté à des redondances compliquant le choix.

Il convient donc de limiter le nombre de services fournis, autrement dit de maximiser le volume de la taxonomie couvert par chaque service. Ceci peut être atteint en produisant des services ayant une qualité de service élevée, et capables de robustesse dans des environnements variés. Réciproquement, les intersections entre services compliquent la tâche de Laurence du fait des redondances : il faut donc veiller également à limiter les intersections.

Notons qu'une approche *ad hoc* améliore la réutilisabilité (pour Laurence), puisque l'API sera réduite et les services bien connus. Elle ne nuit néanmoins pas à l'extensibilité de la boîte à outils (pour Patrick et Stan). En effet, seule la couche « services », de plus haut niveau, est *ad hoc* : les briques constituantes des services, elles, peuvent être génériques et réutilisables.

5.1.2 Recouvrement fonctionnel

Nous voulons proposer un ensemble de services permettant de satisfaire les besoins exprimés dans 3, c'est-à-dire parvenir à un recouvrement fonctionnel du besoin. L'objectif de cette section est de présenter notre démarche pour parvenir à ce recouvrement.

Nous décrivons ci-après deux démarches classiques pour parvenir à un recouvrement : la démarche dite *top-down* et la démarche *bottom-up*. La démarche naturelle dans le domaine de l'interaction Homme-machine est *top-down*. Cependant, dans notre contexte, suivre strictement l'approche top-down conduit à un échec. Ceci justifie une démarche mixte : empirique et itérative, notre démarche permet de parvenir à un recouvrement moyennant un compromis sur le contrat passé entre Laurence et Patrick.

5.1.2.1 Deux démarches : *top-down* et *bottom-up*

La démarche *top-down* est centrée sur le besoin. Elle consiste à exprimer un besoin de service perceptuel pour l'interaction dans la taxonomie (rôle de Laurence), spécifier son interface (rôle de Patrick), puis implémenter les technologies de vision nécessaires (rôle de Stanislas). Elle découle naturellement de l'approche spécifique ; elle peut être répétée pour chaque besoin identifié.

La démarche *bottom-up* est centrée sur la technologie. Elle consiste à choisir des techniques de vision adaptées à une portion de l'union des besoins, les implémenter, proposer une interface pour encapsuler la fonctionnalité offerte, enfin évaluer le volume de la taxonomie remplit par le service. Elle peut être répétée jusqu'à recouvrement de l'ensemble des besoins. Utilisée seule, c'est une piste pour obtenir une interface générique.

5.1.2.2 Échec de la démarche *top-down*

Comme nous l'avons expliqué en 2.2 (page 28), certains besoins sont tout simplement trop difficiles à résoudre en vision : soit parce que la communauté scientifique n'a pas encore de solution, soit parce que le matériel n'est pas encore adapté (puissance de calcul, fréquence ou définition des capteurs, etc.). La démarche *top-down* sera donc souvent vouée à l'échec, car la réalisation d'un service correspondant précisement au besoin est impossible.

Il est délicat d'illustrer ceci, car de tels échecs sont rarement documentés ou publiés ; nous utilisons donc un exemple issu de notre équipe de recherche. Dans le cas de *TROC*, que nous avons décrit page 38, les concepteurs ont expérimenté avec la vision pour la géolocalisation des joueurs (en utilisant *ARToolkit*, c.f. page 71). Cette piste a cependant été abandonnée, car elle impose d'équiper fortement l'environnement (en y plaçant de nombreux marqueurs), et aurait une latence trop élevée (1 s).

5.1.2.3 Le compromis *bottom-up* ou biais technologique

Rappelons que nous supposons dans ce document que Laurence a choisi d'utiliser la vision. Dans le cas où la démarche *top-down* est un échec, Laurence est donc contrainte de faire des concessions sur le besoin qu'elle a formulé : par exemple en acceptant une latence plus élevée ou un environnement d'usage plus contraint.

Nous nommons ce phénomène le *biais technologique*, car Laurence est poussée à concevoir un système interactif d'après la technologie disponible. Cette démarche est mal considérée en interaction Homme-machine, car toute évaluation utilisateur du système interactif est alors biaisée par les limitations de la technologie utilisée. Nous considérons cependant qu'elle est indispensable, car elle permet d'explorer de nouveaux domaines d'interaction, et qu'elle est d'ailleurs largement employée sous la forme des techniques magicien d'Oz, le magicien remplissant alors le rôle de l'*enabling technology*. Le point essentiel est que, après avoir biaisé sa conception, le système de Laurence reste interessant d'un point de vue utilisateur.

En vision, on peut citer comme exemple de biais technologique le système TAFFI proposé par [Wilson, 2006], représenté sur la figure 5.1. Il s'agit d'un système permettant l'interaction bimanuelle avec une interface graphique par l'intermédiaire de gestes de « pincement » (le pouce et l'index en contact délimitent un ovale), détectés par une caméra. Andrew Wilson joue ici d'abord le rôle de Laurence, en définissant une nouvelle technique d'interaction et le besoin de perception afférent (détection et suivi des « ovales de pincement », les mains étant posées sur la surface du bureau, avec une latence basse et une précision millimétrique), puis celui de Patrick en définissant une API pour un service correspondant. Jusque ici, la démarche *top-down* est employée.

Adoptant ensuite le rôle de Stanislas, il constate (d'après nous) que l'environnement doit être plus contraint pour pourvoir implémenter le service perceptif. Finalement, le système ne fonctionne que si (*a*) l'arrière-plan est plus sombre que les mains, ce qui est le cas (par exemple) quand l'utilisateur effectue ses gestes au-dessus d'un clavier sombre, et (*b*) la scène est dénuée d'objets parasites.

Finalement, le système obtenu est suffisemment utilisable pour permettre son auteur d'explorer son utilisation pour l'interaction (émulation de la souris, de la molette, interaction bimanuelle).

5.1.2.4 Synthèse : une démarche empirique et itérative

Isolées, les méthodes *top-down* et *bottom-up* ne permettent pas de produire des services. Nous proposons d'adopter une démarche mixte, empirique, et itérative pour atteindre un recouvrement des besoins.

Figure 5.1 . Le système *TAFFI.*

D'après [Wilson, 2006]. Littéralement, TAFFI (*Thumb And Fore Finger Interface*) signifie « interface au pouce et index ». Une caméra posée sur l'écran observe le clavier de la station de travail. Les mains de l'utilisateur sont segmentées et considérant que toutes les régions plus claires qu'une image de référence font partie des mains.

Par exemple, nous verrons plus bas que nous avons choisi de recouvrir les besoins **S7** à **S11** par deux services, *FingerTracker* et *SensitiveWidgets*. La démarche, visible dans l'historique du déploiement des services (section 7.1.1 page 145), a été d'accroître le volume de la taxonomie recouvert par le *FingerTracker* autant que possible (en améliorant la qualité de service, et en étendant l'environnement toléré), afin de remplir « au mieux » le contrat. Nous verrons qu'il n'a pas été possible de satisfaire parfaitement le contrat (c.f. évaluation, page 164) : Laurence a adapté son système interactif aux limites du *Tracker*, en particulier en termes d'autonomie. Les limitations nous ont également imposé de créer un nouveau service *SensitiveWidgets* pour satisfaire **S7**. Ceci nous permet de répondre aux contraintes d'environnement et de qualité de services tout en satisfaisant fonctionnelement les besoins. Ainsi les *SensitiveWidgets*, quoique fonctionnellement plus limités, sont robuste à un environnement plus large (en particulier, avec un éclairage quelconque).

Ces deux services se recouvrent en partie ; pour certains besoins l'un et l'autre sont valides. Nous verrons par exemple que les boutons de l'interfacce *DoodleDraw* (présenté page 153) auraient pu être implémentés avec les *SensitiveWidgets* au lieu du *FingerTracker*.

Notre démarche générale pour la conception et le développement de *gmlVision* consiste à adopter successivement les rôles de Laurence, Patrick, et Stanislas, en répétant les phases *top-down* et *bottom-up*. Le cycle de développement que nous adoptons est donc le suivant :

1. Laurence formule un besoin non encore satisfait par *gmlVision*, dans le cadre de notre taxonomie, c'est-à-dire donne à Patrick un contrat à remplir ;
2. Patrick conçoit un service adapté et définit les primitives de vision par ordinateur existantes (éventuellement parmi des primitives existantes) ;
3. Stanislas implémente les algorithmes de vision de manière à fournir une qualité de service la plus élevée posible (i.e. couvrant un volume le plus vaste posible dans la taxonomie) ;
4. Patrick reformule la spécification du service, c'est-à-dire renégocie le contrat avec Laurence, après avoir vérifié l'utilisabilité du service ;

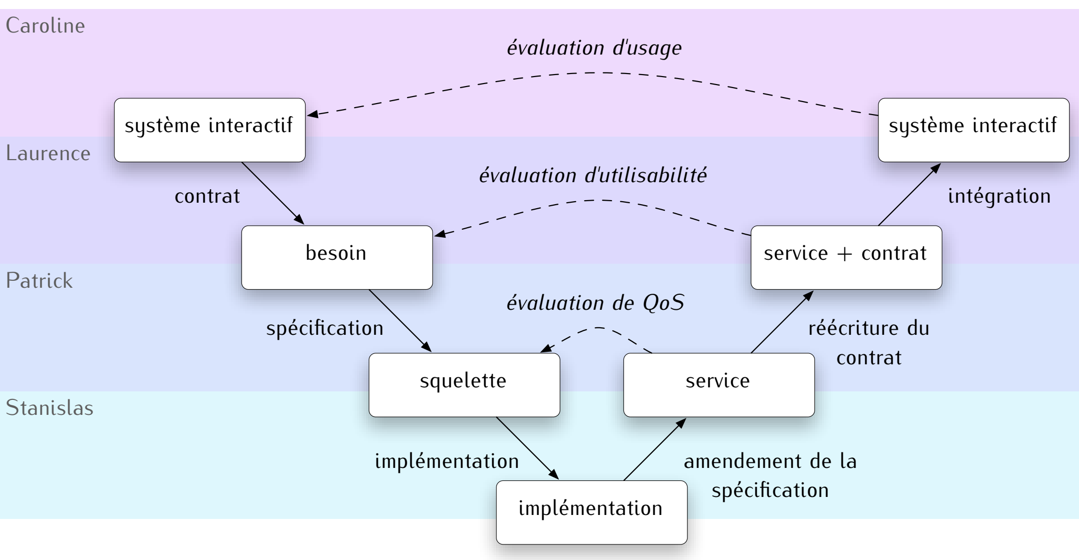


Figure 5.2 . Cycle de développement adopté pour *gmlVision*.

Le cycle de conception et développement que nous avons choisi est une adaptation du « modèle en V » traditionnel [IABG, 1992]. La branche gauche correspond à la démarche *top-down*, et la branchs droite à la démarche *bottom-up*.

5. Laurence modifie la conception de son système interactif au vu du nouveau contrat, et y intègre le service.

Ce cycle est synthétisé sur la figure 5.2 ci-contre. Il est répété pour chaque besoin, jusqu'à obtenir le meilleur recouvrement possible de l'ensemble des besoins de Laurence. Nous définissons un recouvrement optimal comme un recouvrement qui

– maximise le recouvrement de chaque besoin (i.e. où le contrat effectif est le plus proche possible du besoin exprimé), et
– minimise le nombre total de services fournis et les intersections entre services.

Nous décrivons l'ensemble recouvrant obtenu dans la section 5.3 de ce chapitre.

L'attention du lecteur est portée sur le fait que la démarche décrite ici a émergé au cours du développement de la boîte à outils *gmlVision* : il n'a donc pas été appliqué tel quel à tous les services dont nous décrivons la conception dans la troisième partie de ce chapitre. Par souci de clarté, nous prendrons soin d'indiquer l'« état d'avancement » du cycle de développement de chaque service, sous la forme d'un des pictogramme indiquant lesquelles des cinq étapes ont été remplies :

5.2 Architecture

Dans cette section, nous présentons la conception de l'architecture et de l'interface utilisateur de *gmlVision*. Nous décrivons tout d'abord les choix technologiques de haut niveau qui semblent pertinents (à l'exclusion des choix techniques permettant d'implémenter l'interface). Nous présentons ensuite l'interface dite *externe*, orientée service, qui s'adresse à Laurence, et dans une moindre mesure à Patrick. Enfin, nous décrivons l'interface *interne*, à composants, qui s'adresse à Stanislas et à Patrick.

5.2.1 Choix de conception

Nous avons rapporté au chapitre précédent les critères de Myers permettant de qualifier une bonne boîte à outils logicielle (seuil fonctionnel, etc). D'après Myers, le respect de ces critères justifie le succès de certains choix technologiques qu'il présente ensuite [Myers et al., 2000]. Nous rapportons ici ces choix technologiques, validés par des années de « sélection naturelle », et nous nous attacherons à les respecter dans la conception de *gmlVision*.

langages et boîtes à outils à événements. Le paradigme a événements, dont nous avons vu l'intérêt pour Laurence au chapitre précédent (c.f. figure 4.3 page 82), est particulièrement bien adapté à la perception de l'activité humaine, typiquement imprévisible. Il en permet le traitement de manière asynchrone. Parmi les boîtes à outils exploitant ce paradigme, on peut citer Tcl/Tk [Ousterhout, 1994] et GLUT [Kilgard, 1996]. Il n'existe pas, à notre connaissance, de boîte à outils pour interfaces perceptives utilisant ce paradigme.

outils de conception graphiques et interactifs. À l'instar des *interface builders* (Apple HyperCard, NeXT Interface Builder), qui permettent la conception et l'implémentation d'interfaces utilisateur de manière graphique, les domaines du traitement de signal et de flux video en particulier disposent d'outils graphiques pour la conception. L'outil Simulink de Matlab, le Quartz Composer d'Apple, ou le *pipeline editor* de Gnome gstreamer permettent la construction d'algorithmes de vision par ordinateur en assemblant graphiquement des boîtes qui représentent des primitives de traitement d'image ou d'autres algorithmes. Ce type d'outil permet à Stan un cycle de conception, développement, et test plus rapide. Ce raisonnement est également applicable à Patrick, pour l'intégration de services multiples.

programmation orientée objet et systèmes à composants. La combinaison de cette technique de programmation et de cette approche structurelle permet de modulariser une application tout en préservant la richesse fonctionnelle de la boîte à outils. Elles permettent de minimiser l'effort de développement (le seuil fonctionnel) en augmentant la réutilisabilité (le plafond fonctionnel). En encadrant le développeur pour la création de modules, elle fournissent également un « chemin de moindre résistance ».

langages de scripts interprêtés. Sans perdre le pouvoir expressif des langages de plus bas niveau (C, C++, Java), les langages interprêtés masquent des aspect matériels (gestion de la mémoire par exemple) : ils permettent donc naturellement de produire des programmes plus stables Ils fournissent plus de dynamisme à la conception et à l'exécution. Ils permettenr un développement plus rapide, en particulier en minimisant le cycle programmation--compilation--test unitaire et en réduisant les erreurs de programmations possibles. Parmi ces langages, on peut citer Python [van Rossum, 2006], Tcl [Ousterhout, 1994], ou Smalltalk [Goldberg et Robson, 1983].

5.2.2 Interface externe orientée service

Nous présentons ici la conception de l'interface de *gmlVision* ; c'est-à-dire que nous founissons une description de haut niveau de l'API, sans rentrer dans les détails de l'implémentation (langages, structures de données, ou algorithmes). Ces détails seront fournis au chapitre suivant (*Mise en oeuvre*, page 117).

Pour manipuler des services perceptifs, l'activité de Laurence s'articule en trois points. Il lui est tout d'abord nécessaire de détecter ou découvrir le service d'intérêt présent dans l'environnement interactif, et de s'y connecter. Elle doit ensuite recevoir et interprêter les messages reçus du service, qui contiennent l'information de perception. Enfin, elle devra occasionnellement contrôler, inspecter, ou configurer un service perceptif.

Nous décrivons ci-dessous les solutions apportés dans *gmlVision* à ces problèmes. Elles sont issues pour partie directement des principes énoncés au chapitre précédent, pour partie des leçons tirés des itérations de développement et de déploiement des middlewares bâtis sur ces principes.

5.2.2.1 Canaux de communication multiples

Au cours des cycles itératifs de développement de *gmlVision* et des intergiciels associés est apparu le besoin d'aggréger plusieurs services conceptuellement proches en une seule entité. Deux cas existent :

- un capteur physique fournit plusieurs services identiques (i.e. instances de la même classe) ou liés (par exemple synchronisés). C'est la cas d'une grille de microphone (qui fournit de multiples services *microphone* synchronisés). En vision par ordinateur, un exemple est celui d'une caméra stéréoscopique, qui produit une carte de profondeur, une image rectifiée, et les deux vues.
- un même logiciel fournit des variantes du même service. Par exemple, le service chargé d'abstraire une caméra video (`gml.input.video`) fournit un service « flux video » (chaque image capturée est envoyée au client) et un service « appareil photo » (une image sont envoyées à la demande). Un autre exemple est le service de suivi de doigts (`gml.tracker.finger`), qui fournit en outre le suivi des doigts, un service de détection simple (sans suivi) et un service d'émulation de clics (détection de la pause d'un doigt).

L'approche la plus élégante serait de fournir, à l'exécution, une instance de service pour chacun des sous services. Cependant, ceci élève le seuil fonctionnel pour Patrick et pour Laurence. Patrick devra fournir un effort plus grand pour instancier un plus

grand nombre de services. Laurence devra fournir un effort plus grand pour découvrir l'ensemble des services liés -- dans le dernier exemple, deux cycles découverte-connexion au service seront nécessaires si elle doit accéder au suivi et à la détection. Il est donc préférable d'aggréger ces « sous-services ».

Nous choisissons de matérialiser ces sous-services sous la forme de *canaux de communication* multiples. Chaque service possède donc un ou plusieurs canaux. Un client se connecte à un canal particulier d'un service pour accéder à sa fonctionnalité. Un des canaux est le canal par défaut ; c'est celui qui fournit le sous-service le plus usuel.

Le choix de canaux multiples présente un écueil à éviter pour Patrick : celui du *god service* (par analogie avec les *god objects* [Smith et Williams, 2000]). Le risque est la création de super-services exposant dans de multiples canaux des services sans rapport, annulant ainsi les gains structurels de l'approche à service -- on retournerait alors vers une approche monolithique. Dans les termes de Myers, nous faisons le compromis d'améliorer le seuil et le plafond fonctionnel en autorisant plusieurs canaux de communication par service au prix d'une déterioration du chemin de moindre résistance. Nous ne proposons pas de mécanisme pour éviter les *god services* : nous laissons cette responsabilité à Patrick.

5.2.2.2 Nommage, découverte et détection de services

Il est possible de spécifier, sans présumer des choix implémentationnels, que chaque canal sera accessible par le biais d'une connection TCP. Il s'agit en effet de l'unique mécanisme de transport universellement accessible (i.e. depuis toute plate-forme logicielle). Un canal est donc défini par un couple adresse IP, numéro de port (ou nom d'hôte, numéro de port).

Il est naturel pour Laurence d'accéder à la fonction d'un service de manière symbolique plutôt qu'en spécifiant le mécanisme de bas niveau par lequel s'effectue la connexion entre son application et le service. En particulier, l'adresse IP et le numéro de port peuvent ne pas être connus avant l'exécution du service. Nous proposons donc que l'interface d'accès aux services permette à Laurence de *nommer* le service à utiliser, plutôt que de spécifier une adresse de canal.

Au moment du développement de l'application interactive, Laurence ne peut présumer de la localisation du service perceptif à l'exécution (i.e. de sa présence dans le même processus logiciel ou sur la même machine), ni même de sa présence. Cette décision permet de préserver la flexibilité et la fiabilité du système interactif dans son ensemble et participe à satisfaire le requis d'**isolation** (c.f. page 83). Par conséquent, il est pertinent de proposer un mécanisme permettant de détecter un service adapté, puis de s'y connecter.

Il est possible que l'environnement logiciel fournisse plusieurs instances du même service. On peut par exemple imaginer le service de suivi de jetons de la *Table Magique* fonctionnant en parallèle sur une table et sur un tableau blanc, dans la même pièce. Les deux instances du service peuvent même potentiellement être exécutées sur la même machine physique. Laurence doit donc pouvoir spécifier la *classe* de service qui satisfait son besoin (ceci au moment de la conception), et choisir l'*instance* particulière à laquelle elle s'interesse (à l'exécution).

Nous proposons donc d'identifier tout service par sa *classe*, une chaîne de caractère identifiant la portée du service dans la taxonomie (c'est-à-dire son contrat) et un *nom*, une chaîne de caractère identifiant l'instance de manière unique à un instant donné. La classe et le nom d'une instance ne varient pas durant son existence. De même, les canaux d'un service sont identifiés par un nom (une chaîne de caractère) unique pour une classe de service donnée.

Néanmoins, dans le cas le plus courant, l'environnement logiciel contiendra une unique instance de service perceptif et une unique application cliente, toute deux exécutées sur la même machine. L'implémentation devra donc satisfaire ce cas de base le plus

simplement possible, en particulier sans faire intervenir les notions de classe et de nom de service.

Par souci d'obtenir un seuil bas, l'information minimale (adresse IP, numéro de port) doit être suffisante pour que Laurence se connecte à un service : les mécanismes de nommage, découverte et détection de service doivent être optionnels.

5.2.2.3 Évènements perceptuels

Nous avons montré qu'une approche à événements était préférable pour permettre à Laurence d'interagir avec un service perceptif. Un événement est un objet simple, c'est-à-dire dépourvu de méthodes au sens de la programmation orientée objet : il s'agit simplement d'un conteneur d'information. En d'autres termes, il s'agira d'une structure de données de type dictionnaire en Python, tableau associatif en Tcl ou Perl, ou d'un `struct` en C ou en C++. Il possède au moins deux attributs : son *nom*, un identifiant textuel de sa classe ; et un *timestamp*, représentant l'instant où l'événement perceptuel a eu lieu dans le monde physique.

Pour un suivi visuel des propriétés d'un agent, trois classes d'événements existent :

<Appear> Un agent est détecté pour la prémière fois, en particulier parce qu'il entre dans le champ de la caméra, qu'il vient de devenir visible, ou que le système perceptif vient de démarrer ou de reprendre sur panne.

<Update> Une ou plusieurs des propriétés d'intérêt de l'agent a évolué de manière significative (i.e. au-delà du seuil de stabilité statique).

<Disappear> L'agent a cessé d'être visible. Il peut avoir été masqué ou occulté pendant une période prolongée, être sorti du champ de la caméra, ou ne plus pouvoir être détecté pour toute autre raison.

Notons ici n et p les bornes inférieure et supérieure du nombre d'agents interactifs tolérés par un service (pour mémoire, il s'agit de l'un des axes de la taxonomie). Lorsque $n = p = 1$, `<Appear>` et `<Disappear>` ne sont pas pertinents : l'agent est supposément toujours présent, et seul `<Update>` est pertinent. Ce cas est à mettre en parallèle avec celui de la souris. Lorsque $p > 1$, il faut identifier de quel agent provient l'événement ; on ajouter à cet effet un champ identifiant l'agent.

Certains services doivent par nature offrir à Laurence une interface transactionnelle, c'est-à-dire de type invocation distante de méthode (RMI). Nous verrons plus bas que c'est le cas du service perceptif de capture d'aspect (`gml.grabber.surface`) et de certains services aidant à la construction d'applications (`gml.calibrator.surface` et `gml.display`). Comme cette interface doit également être asynchrone, nous proposons de conserver l'approche à événements, en introduisant des événements requête et des événements réponse. Par exemple, pour la capture d'aspect, les événements seront `<ImageQuery>` et `<ImageResponse>`, associés respectivement à une description géométrique de la zone à capturer, et à l'image correpondante. Les événements utilisés pour la RMI sont également associés à un identifiant de requête, choisi par l'emetteur de la requête, afin de dinstinguer le cas échéant des requêtes concurrentes. Laurence, client de *gmlVision*, pourra donc être emmettrice d'événements et pas uniquement consommatrice.

5.2.2.4 Représentation des événements

Il est nécessaire de représenter les événements de la manière la plus simple possible. En effet, une représentation complexe :

- a un effet négatif sur la latence : il sera nécessaire de sérialiser et déserialiser chaque événement, ce qui est d'autant plus coûteux que la représentation est complexe ;
- élève le seuil fonctionnel : Laurence (et Patrick) devront manipuler un ensemble de fonctions dédiées à la manipulation de la représentation. En particulier, détériore l'observabilité des services, en ne fournisant aucun moyen simple d'inspecter leurs sorties ;

– déteriore l'interopérabilité : une représentation complexe impose à Laurence d'utiliser des outils qui ne sont pas forcément disponibles dans son environnement de développement.

Il est donc préférable de ne pas employer de représentation verbeuse ou complexe, comme un sous-langage de XML, pour représenter les événements. Comme les événements perceptuels sont des structures de données très simples, nous suggérons d'utiliser une représentation textuelle (ASCII).

Sans présumer des choix d'implémentation, nous pouvons déjà supposer que les événements seront représentés comme une succession de messages textuels de la forme :

```
<Update> id=0x12345678 x=0123.45 y=0678.90
```

pour un simple suivi sur une surface interactive, avec identité maintenue, comme `gml.tracker.color`. Ici, les attributs de l'événements sont *id*, l'identité de l'agent interactif, et x et y, les coordonnées euclidiennes de l'agent. Pour les services où les données liées aux événements sont volumineuses (comme la famille `gml.grabber.*`), il conviendra de choisir une représentation compatible avec ce choix (par exemple en compressant les images, ou en utilisant un mécanisme de transport offrant un débit élevé ; c.f. 5.2.3.4 page 100).

5.2.2.5 Contrôle et configuration des services

La plupart des services peuvent être configurés, en particulier pour modifier la qualité de service fournie. Par exemple, l'algorithme utilisé dans `gml.tracker.striplets` (detection d'occlusion par un doigt) peut être rendu plus robuste aux occlusions intempestives et aux variations de l'éclairage en augmentant sa latence. De la même manière, la latence de `gml.tracker.finger` peut être baissée, au détriment de sa précision, en diminuant la résolution de l'image d'entrée (et réciproquement). Cette configuration peut être nécessaire pour permettre à Laurence d'obtenir un service parfaitement adapté à son besoin : en configurant le service, elle peut ajuster le contrat qu'il remplit.

À cet effet, les services de *gmlVision* possèdent un canal de communication particulier, dit *canal de contrôle*, qui permet ce type de configuration. Ces mécanismes sont décrits plus en détails au chapitre suivant (section 6.1.1 page 118) et dans l'annexe A page 177.

5.2.2.6 Conclusions

L'interface conçue pour Laurence est volontairement minimale. Par souci de maintenir un seuil fonctionnel bas, elle doit permettre à Laurence de se connecter à un service et de recevoir des événements perceptuels en fournissant un minimum d'efforts : établir une connection TCP avec le service, et traiter les événements présentés sous forme de texte ASCII.

Néanmoins, l'utilisateur a la possibilité de découvrir dynamiquement les services perceptuels, de les inspecter, et de les configurer si nécessaire. Ceci permet d'élever le plafond fonctionnel de *gmlVision* en permettant en particulier son intégration dans des environnements perceptifs ou logiciels divers.

5.2.3 Interface interne et support au développement des services

Une approche à services ou à composants est acceptable lors le coût ajouté par le *framework* (cadriciel) utilisé est négligeable en termes de temps, quantité et qualité de développement d'un service, et de performances ou qualité de service. Une telle approche est bénéfique à l'utilisabilité de la boîte à outils si elle facilite en outre le développement et/ou permet d'atteindre de meilleures performances.

Ici, nous proposons une architecture interne de la boîte à outils, centrée sur les besoins de Stanislas et Patrick, permettant le developpement rapide de services en respectant le requis présentés au chapitre précédent.

5.2.3.1 Primitives de traitement d'image et de vision par ordinateur

La plupart des primitives de traitement d'image sont des operations qui utilisent une ou plusieurs images pour produire une ou plusieurs images. Parmi les plus classiques, on peut citer les opérations suivantes :

- filtrage d'image : convolutions, opérations morphologiques, détection de sur-exposition ;
- transformations d'espaces de couleur, linéaires ou non : RGB, espaces perceptuels (HSV, YCrCb) ;
- combinaison d'images : distance entre images, alpha-blending, chroma-keying ;
- transformations géométriques : décimation, mise à l'échelle avec filtrage bilinéaire ou bicubique, transformation projective ;
- changements d'espace : transformée de Hough, transformée de Fourier ;

Certaines primitives produisent d'autres formes d'information, de dimension plus faible : ces sont elles qui extraient effectivement l'information d'une ou plusieurs images. Ces primitives d'« intégration » sont moins nombreuses et bien connues. Par exemple :

- mesures : statisques globales (calcul d'entropie, de moments d'ordre 0 et 1), histogrammes ;
- opérations structurantes : analyse en composantes connexes, chaînage de pixels (utilisé dans pour l'opération de détection de contours de Canny, ou les opérations de polygonalisation de contours), blobs [R. Kauth, 1977], différentes segmentations (region-growing, watershed).

Afin de rendre la boîte à outils utilisable pour Stan, nous choisissons d'orienter la conception de cette partie d'après les trois critères suivants :

performance. Les opérations traitant des images sont les plus coûteuses en temps de calcul. Ces sont donc celles qui ont le plus d'impact sur la latence du service dont elles sont partie. Il sera donc généralement nécessaire de les implémenter dans un langage de bas niveau — C ou assembleur.

seuil fonctionnel faible. Afin de minimiser l'effort d'appentissage fourni par Stanislas, l'ensemble des primitives de traitement d'image devra posséder la même interface, exposée dans le langage interprêté. En outre, *gmlVision* doit fournir un éventail varié des primitives les plus classiquement utilisées en vision par ordinateur. Le réalisation de ces primitives se fera naturellement au cours de la réalisation des services décrits dans la section suivante (5.3 page 102).

plafond fonctionnel élevé. La boîte à outils doit être réutilisable et extensible. Il est donc necessaire de fournir un chemin de moindre résistance pour l'implémentation des primitives développées et utilisées par Stanislas.

Notre objectif principal étant de satisfaire Laurence, nous ne détaillons pas ici plus avant la reflexion centrée utilisateur menant à la conception de la couche « imagerie » de *gmlVision*. Dans l'annexe B (page 187) nous décrivons les mécanismes fournis par notre boîte à outils permettant de satisfaire ces contraintes : les opérations de traitement d'images sont décrits dans un langage *ad hoc* à partir duquel le code et la documentation sont générés. Ceci permet de garantir l'uniformité de l'interface et de contribuer à satisfaire le requis de performance.

5.2.3.2 Composants légers

Nous avons conclu au chapitre précédent qu'une architecture à composants est intéressante pour la construction d'un service particulier. Nous avons également constaté que les architectures à composants existantes étaient peu satisfaisantes d'un point de vus centré utilisateur : elle ont un seuil fonctionnel élevé et ne sont pas assez performantes pour nos besoin.

Nous proposons donc d'utiliser une architecture concrète à composants « légers ». Cette architeture est contrainte par les flux de données qui circulent entre composants dans un système de vision par ordinateur. Ces flux sont asynchones (il s'agit

d'événements discrets), volumineux (les données associées peuvent être des images), non linéaires (un flux peut être distribué à plusieurs composants), et dynamiques (la connexion des composants peut être modifiée à l'exécution).

Afin de satisfaire ces contraintes, nous proposons de définir une classe `EventSource` dont les classes dérivées seront les composants « légers ». Un composant publie les événements qu'il peut emettre ; tout autre composant peut s'abonner ou être abonné à un ou plusieurs événements. `EventSource` fournit les méthodes suivantes :

constructor (***event list***)
: L'ensemble des événements émis par cet objet est spécifié à l'instantiation.

bind ($\varnothing \to$ ***event list***)
: Méthode d'inspection : renvoie la liste des événements émis par l'objet.

bind (***event***, ***callback*** $\to \varnothing$)
: Ajoute *callback* à la liste des fonctions à appeler lorsque l'objet émet l'événement *event*. Généralement *callback* est une méthode liée, i.e. un couple objet, méthode.

unbind (***event***, ***callback*** $\to \varnothing$)
: Réciproque de **bind** : supprime un *callback*.

invoke (***event***, ***data*** $\to \varnothing$)
: Émet l'événement *event*, avec les données associées *data* ; c'est-à-dire, invoque tous les *callbacks* précedemment associés à *event* en utilisant **bind**.

En utilisant cette interface, tout objet devient un composant capable de produire et recevoir des événements. Nous ne fixons pas de contraintes sur l'instanciation des composants. Ils seront construits et assemblés par un superviseur, ou objet intégrateur, en général l'objet qui abstrait le service perceptif.

5.2.3.3 Conception de l'architecture à services

Pour permettre à Patrick de construire des services la boîte à outils doit fournir un ensemble de primitives matérialisant les concepts de service et de canal décrits plus haut dans ce chapitre. Puisque nous choisissons d'adopter le paradigme de programmation orientée objet ces concepts deviennent des classes `Service` et `Channel`. Comme la communication avec d'autres services ou avec le client applicatif est asynchrone, `Channel` sera naturellement un composant léger dérivé de `EventSource`.

L'activité principale de Patrick est la création d'un service perceptif : il s'agit de définir sa classe, un canal d'entrée, un canal de sortie, et une fonction traitant les événements d'entrée pour produire les événements de sortie. Nous offrons un chemin de moindre résistance pour cette activité.

Dans le cas général, la tâche de Patrick est la suivante :

- définir une classe dérivée de la classe `Service` pour implémenter un service particulier
- surcharger le constructeur de `Service` pour spécifier la classe du service et instancier un canal d'entrée et un canal de sortie, et définir les autres structures de données propres au service ;
- définir une méthode `onReceive` (par exemple) qui traite les événements du canal d'entrée et envoie les événements sur le canal de sortie.

La boîte à outils doit supporter la réalisation de cette tâche de manière compacte en utilisant des structures de données les plus proches possibles des concepts manipulés. Nous verrons au chapitre suivant, ainsi que dans l'annexe A, que l'implémentation de `gmlBIP`, le middleware orienté service utilisé par *gmlVision*, suit très précisément les règles de conception présentées ici (c.f. figure A.1 page 186).

Patrick doit éventuellement pouvoir dériver la classe `Channel` pour implémenter des sérialisations spécifiques ou des transformations de données ; la bibliothèque fournit des spécialisations de `Channel` pour les tâches classiques. En général les événements

d'entrée sont un flux d'images provenant d'une caméra. Nous avons identifiés ces spécialisations comme étant celles requises de manière répétitives lors de la conception et de l'implémentation de *gmlVision*. Nous décrivons les classes correspondantes dans les paragraphes suivants.

5.2.3.4 Transfert d'images entre services

L'ensemble des services perceptifs que nous construisons ont pour entrée une source d'image et parfois plusieurs. En général cette source est l'abstraction d'une caméra video. Parfois il s'agit d'un autre service produisant un flux video ou des images ponctuelles. Nous verrons plus loin dans ce chapitre que nous devons isoler l'abstraction d'une caméra dans un service dédié : `gml.grabber.camera` (discussion page 111).

Un lien véhiculant des images est soumis a des contraintes particulières :

- il doit être performant : le volume de données étant élevé, il peut devenir une source de latence non négligeable en particulier si les images sont sérialisées ;
- il doit être utilisable par Laurence sans faire intervenir d'intergiciel spécifique (afin de ne pas élevé le seuil fonctionnel).

Un solution est de fournir des canaux spécialisés capables de transferts performants entre services (utilisant *gmlVision* et son intergiciel) et de transferts simplifiés avec un autre client (une application interactive écrite par Laurence). Lors du transfert simplifié les images seront sérialisées sous la forme de fichiers image dans un des formats standards (JPEG, TIFF ou PNG selon le type d'image transmise) afin de préserver l'interopérabilité.

Pour Patrick, une implémentation des deux extrémités de ce canal spécialisé utilisable dans un service doit être fournie. Nous nommons ces classes dérivées de `Channel`, `ImageSourceChannel` (canal d'un service émettant des images) et `ImageSinkChannel` (canal recevant des images).

5.2.3.5 Mécanismes d'adaptation fonctionnelle

En suivant la démarche que nous avons adopté il est courant qu'un service fournisse une fonction plus complète que le besoin de Laurence. C'est le cas des services de suivi : *DoodleDraw* par exemple est implémenté à l'aide de `gml.tracker.finger`. L'application a besoin d'un suivi de *0 à 2* agents mais le service fournit un suivi de *0 à 20* agents. Un besoin apparaît alors : adapter les services au besoin précis en les restreignant fonctionnellement. Ce besoin est d'ailleurs apparu de manière récurrente lors des déploiements de *gmlVision* (c.f. chapitre 7 page 143).

Les besoins identifiés, que les utilisateurs ont implémentés par filtrage des événements issus des services, sont les suivants :

- réduction du nombre d'agents suivi : conversion d'un service dont le nombre d'agents interactif est $0..n$ en $0..p$ ($p < n$) ;
- réstriction de la vue : restriction à une région d'intérêt ;
- réduction de la précision : restriction des informations de position à une grille (*clamping*), comme dans l'application *Caretta* ;
- augmentation de la latence : détection de pause (*dwelling*) pour les services de suivi, afin d'émuler un clic de souris sur une interface tactile ou tangible.

Comme notre objectif est de fournir à Laurence un service adapté à son besoin, nous devons fournir à Patrick des outils pour implémenter ces restrictions. Les alternatives envisagées sont : soit de fournir des services « adaptateurs » capables de filtrer un flux d'événements, soit fournir des composants équivalents à intégrer dans les services.

Pour Laurence, fournir de nouveaux services n'est pas adapté : il faudrait alors démultiplier son effort fourni pour instancier les services et les connecter. Ces services « adaptateur » ne seraient d'ailleurs pas des services perceptifs ce qui pourrait entraîner la confusion de l'utilisateur. La première solution est donc plus satisfaisante. Afin de ne pas élever le seuil fonctionnel nous proposons de fournir les fonctions de filtrage

sous la forme de sous-classes de `Channel` : le service perceptif n'est pas modifié, c'est le canal de communication qui est chargé du filtrage des événements, les modalités du filtrage étant négociées à la connexion.

Du point de vue de Patrick, l'API n'est donc pas modifiée : il convient simplement d'instancier un canal spécifique plutôt que le canal générique. Pour Laurence, il suffit de spécifier les paramètres du filtrage au moment de la connexion au canal. Les quatres besoins de filtrages énoncés ci-dessus sont satisfaits par la classe `TrackerFilterChannel`.

5.2.3.6 Mécanismes d'inspection et de contrôle

Il est nécessaire de rendre observables et contrôlables les paramètres, l'état, et le comportement de tous les composants d'un service afin de permettre à Patrick et à Stan le développement rapide.

Concernant l'observabilité, Stan et Patrick ont besoin de visualiser (graphiquement) des résultats de traitement. Les trois besoins de visualisations rencontrées lors du développement de *gmlVision* sont :

- les images intermédiaires produites par un algorithme de traitement d'image ;
- des valeurs numériques évoluant dans le temps, comme la latence du système ou une mesures de bruit ;
- une représentation des événements perceptuels, éventuellement graphique : par exemple, les représentations d'événements de suivi doivent être superposés à l'image de la caméra.

Nous proposons de fournir aux developpeurs un mécanisme faiblement invasif permettant de rendre observable ces informations. L'instrumentation doit être légère pour Stan et Patrick afin de maintenir un seuil faible, et ne doit pas dégrader les performances du système perceptif. En s'appuyant sur l'architecture à composants légers proposée ci-dessus nous exposons une API de *monitoring* orientée événements. Tout composant peut emettre des événements `<Monitor>` ; un dispositif d'inspection peut alors s'abonner à ces événements et afficher une représentation graphique.

Pour rendre les services contrôlables nous proposons que tous les services et les composants implémentent une API de contrôle uniformisée permettant l'inspection et la modification des paramètres et variables d'état. Cette API se présente sous la forme d'une classe appelée `Configurable`. Elle doit fournir trois méthodes : **addoption** est invoqué dans le constructeur d'un `Service` pour enregistrer un paramètre ; **cget** est l'acceseur, et **configure** le modificateur de paramètre.

Ces deux API permettent à Stan et Patrick de spécifier ce qui est observable et contrôlable dans les composants et les services ; la boîte à outils doit également founir le moyen pour un utilisateur d'observer et de contrôler. *gmlVision* offre donc un composant graphique générique, `ServiceMonitor`. Il est attaché à un service et fournit un rendu des événements `<Monitor>` reçus, ainsi qu'une interface de contrôle s'interfaçant avec `Configurable`.

D'autre part, afin de permettre le contrôle distant d'un service (depuis un autre programme ou une autre machine), ces API de d'inspection contrôle sont exposés par le biais d'un canal spécifique, présent pour tout service, nommé `control`. Ce canal est implémenté par une classe dérivée de `Channel`, `ServiceControlChannel`. Il s'agit du même canal de contrôle que celui visible par Laurence, et décrit plus haut.

5.2.4 Conclusions

Nous venons de présenter un ensemble de choix structurels pour la conception de la boîte à outils *gmlVision*, compatibles avec les requis énoncés au chapitre précédent. Ces choix ont été déterminés en prenant successivement le point de vue des différents utilisateurs de la boîte à outils. Du point de vue de Laurence, *gmlVision* se présente sous la forme d'une fédération de services qu'elle peut instancier ou découvrir dans

l'envorinnement logiciel. Chaque service est une boîte noire pourvue de canaux de communication permettant l'entrée et la sortie d'événements du service, ainsi que sa configuration. Du point de vue de Stan, la boîte à outils fournit une interface unifiée pour construire et/ou assembler des primitives de vision par ordinateur sous la forme de composants légers capables d'emettre des événements fonctionnels et de surveillance (*monitoring*). Enfin, pour Patrick, la boîte à outils facilite l'assemblage de ces composants à l'intérieur d'un squelette de service grâce aux cannaux de communication et *via* les mécanismes événementiels.

En prenant le rôle de Patrick, l'architecture de *gmlVision* est résumée sur la figure 5.3.

5.3 Un ensemble recouvrant de services

Dans cette section, nous présentons l'interface d'un ensemble de services recouvrant les besoins présentés au chapitre 3 en satisfaisant les requis fonctionnels et structurels présentés dans les deux chapitres précédents. Ces services sont pour partie implémentés dans *gmlVision*, boîte à outils dont les aspects technologiques seront détaillés au chapitre suivant (page 117).

Cet ensemble de services est l'aboutissement de notre démarche empirique. Cet ensemble n'est donc pas nécessairement optimal (au sens de l'optimalité défini en 5.1.2.4), mais doit au minimum permettre de remplir notre objectif : réimplémenter les prototypes présents dans l'état de l'art.

Pour chaque service nous décrivons le volume de la taxonomie qu'il satisfait sous la forme d'un cartouche semi-graphique tel que décrit page 61.

Nous structurons ces services en trois partie : les services de suivi, c'est-à-dire les services perceptifs d'intérêt direct pour Laurence ; les services de capture, réutilisés par la plupart des autres services ; enfin les services de « support », qui permettent la configuration automatique ou assistée des autres services au sens de notre typologie de l'autonomie.

5.3.1 Notations

Nous décrivons brièvement la manière dont les services seront présentés. Chaque service est tout d'abord décrit par son cartouche (légende page 61), puis textuellement. Nous donnons le cas échéant des information sur la démarche ayant abouti au recouvrement par ce service. Les événements échangés avec le service sont ensuite décrits : ils sont précédés par le symbole ⇨ pour les événements sortants et ⇦ pour les événements entrants. Suit le nom de l'événement, sous la forme <Couic>, puis les attributs

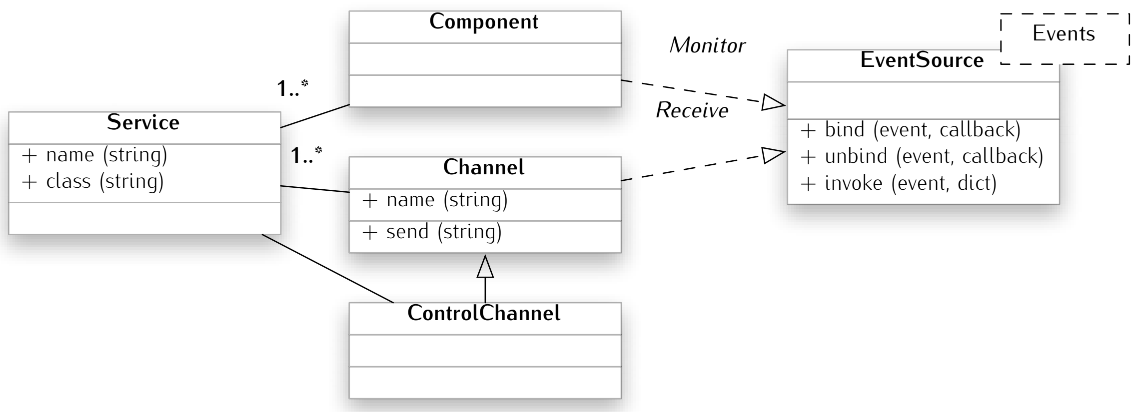


Figure 5.3 . Diagramme de classes UML de l'architecture interne à services de *gmlVision*.

associées. La description des attributs les plus classiques est donnée ci-dessous, nous ne la rappelons donc pas pour chaque événement.

tm (entier)
: instant de l'événement avec une granularité fine (pour *timestamp*).

id (entier)
: identifiant unique ou pseudo-unique de l'agent source de l'événement.

x, *y* (flotants)
: coordonnées euclidiennes, dans les unités et le référentiel de l'interface graphique. Dans le cas où le service fournit explicitement des informations relativement au capteur, indique une position sur l'image caméra.

ox, oy (flottants)
: coordonnées normalisées d'un vecteur orientation.

sx, sy (flottants)
: dimensions en pixels, selon les deux axes principaux d'un objet.

color (entier)
: repérésentation RGB d'une couleur (typiquement sur 24 bits, les 8 bits de poids fort représentant la composante R).

image (données brutes)
: représentation d'une image, encapsulée dans un fichier (typiquement au format TIFF ou JPEG).

$\varnothing$ (vide)
: aucune donnée n'est associée à l'événement.

Tous les événements sortant d'un service sont étiquetés avec *tm*. Cet attribut ne sera donc pas mentionné dans les descriptions des services.

Enfin, sauf si le contraire est spécifié, les services possèdent un unique canal de sortie, qui est le canal par défaut (c.f. 5.2.2.1).

5.3.2 Services de suivi

La notion de *suivi* est prise ici au sens large. Nous définissons le suivi comme la notification discrète de l'évolution dans le temps d'une ou plusieurs propriétés d'un ou plusieurs agents interactifs. Ainsi, la plupart des besoins de perception mis à jour aux chapitre 3 peuvent être considérés comme des besoins de suivi.

Que ce soit pour le suivi de doigts, de jetons, de post-its, ou de *phicons*, tous les services présentent une fonctionnalité similaire et doivent donc présenter la même interface.

L'ensemble des services de suivi est bien entendu extensible (par exemple pour suivre d'autres classes d'agents, ou d'autres propriétés), et on peut imaginer un métaservice de suivi. Nous savons qu'un tel métaservice n'est pas réalisable et ne serait pas utilisable. Néanmoins, le coût de creation d'un « nouveau » suivi ou d'extension d'un service existant est faible : les briques permettant de construire un service sont réutilisables. Nous avons d'ailleurs pu réaliser à faible coût différentes extensions de `gml.tracker.finger` en ajoutant de nouvelles propriétés d'intéret ou en rendant possible sont utilisation dans des environnements différents, créant ainsi de nouveaux services.

5.3.2.1 Remarque préliminaire : utilisation de l'infrarouge

Les sources de lumière artificielle qui n'utilisent pas l'incandescence n'émettent généralement pas dans l'infrarouge : ce serait un gaspillage d'énergie car leur lumière est destinée à un usage par l'Homme et l'Homme ne perçoit pas la lumière infrarouge. En particulier, les écrans cathodiques ou à cristaux liquides, de même que les

lampes « néon », émettent uniquement dans la partie visible du spectre lumineux. Par ailleurs, les videoprojecteurs sont munis de filtres passe-haut qui éliminent le rayonnement infrarouge.

Pour les surfaces interactives, l'utilisation d'une caméra percevant l'infrarouge proche permet donc de s'affranchir en partie des problèmes d'éclairage, donc de couvrir un environnement d'usage plus vaste tout en augmentant la qualité de service. En particulier, il est possible de projeter une interface sur la surface interactive. Il est donc possible d'implémenter des surfaces interactives avec ou sans retour visuel colocalisé (respectivement comme *RoomPlanner* ou *Visual Touchpad*).

L'emploi de l'infrarouge pour la vision impose une contrainte à Laurence : l'investissement dans matériel non usuel, à savoir une source de lumière infrarouge et un recepteur (caméra). Il est possible cependant d'utiliser du matériel usuel à cet effet. Une lampe ordinaire (à incandescence) peut être utilisée comme source infrarouge (elle a un rayonnement de corps noir). Une caméra ordinaire peut être modifiée pour devenir une caméra infrarouge en lui enlevant son filtre passe-haut et en lui ajoutant un filtre passe-bas, car les capteurs CCD et CMOS sont sensibles à l'infrarouge proche.

Néanmoins, les services utilisant l'infrarouge doivent également être capable de fonctionner avec un capteur ordinaire, quitte à fournir une qualité de service moindre. Ainsi, le seuil fonctionnel de la bibliothèque n'est pas augmenté pour Laurence.

5.3.2.2 Interfaces tactiles

Nous proposons de fournir deux services distincts pour satisfaire les besoins de perception pour les interfaces tactiles. En effet, le sous-espace de la taxonomie correspondant à ces besoins est trop vaste : comme énoncé plus haut, il est nécessaire de faire un compromis entre qualité de service et environnment d'usage.

Figure 5.4 . Filtre infrarouge de la caméra video *Unibrain Fire-i 400*.

Cette caméra (comme la plupart des caméras grand public et industrielles) est pourvue d'un filtre éliminant l'infrarouge (teinté en rouge sur l'image de droite). Ce filtre est simplement posé entre le pas optique et le PCB du capteur photographique. Il suffit pour obtenir une caméra infrarouge de le retirer, et de le remplacer par un filter passe-bas, par exemple un morceau de pellicule argentique totalement exposée.

gml.tracker.finger

agent		*qualité de service*		*image & cadrage*		*eclairage*	
doigt		#agents	0 à 20	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm	couleur	infrared	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☒☐☐☐			contexte	☐	variation	☐

Le service `gml.tracker.finger` permet d'implémenter des interfaces tactiles lorsque la qualité de service (en particulier, la précision et la latence) est élevée, au détriment de l'environnement d'usage. Il sera utilisable dans un environnement contrôlé (éclairage quasi-constant, vue fixe). Un tel suivi de doigts et de mains se comporte, du point de vue de Laurence, comme un ensemble de souris.

`gml.tracker.finger` permet le recouvrement des besoins S7, S9, S10. Il permet également un recouvrement partiel de S14 (i.e. avec un environnement d'usage plus contraint). C'est également un support pour le recouvrement de S8 et S11, comme nous le verrons dans la section suivante.

Il produit les événements suivants usuels des services de suivi sur son canal par défaut, *tracking-events* :

- `<Appear>` (*id*, *x*, *y*, *ox*, *oy*)
- `<Update>` (*id*, *x*, *y*, *ox*, *oy*)
- `<Disappear>` (*id*)

Sur le canal *dwelling-events*, il produit des événements correspondant à la pause d'un doigt (événement fréquemment utilisés pour déclencher des actions, à l'instar du clic de la souris). Ils permettent d'implémenter, par exemple, des boutons tactiles :

- `<Dwell>` (*id*, *x*, *y*)

Enfin, il s'agit d'un service coopérant avec Laurence et Caroline (en termes d'autonomie en ligne). En effet, il n'existe pas encore de technologie utilisant la vision par ordinateur permettant de fournir ce service perceptif avec les performances requises pendant une durée prolongée et de manière autonome. Nous verrons au chapitre suivant que la technique employée doit, à l'initialisation, observer une scène « neutre », c'est-à-dire une scène où aucun agent interactif n'est visible et ou aucun mouvement n'a lieu. Le canal *reset* permet cette initialisation :

- `<ResetQuery>` (∅)
- `<Reset>` (∅)
- `<ResetDone>` (∅)

Le premier événement est émis lorsque le service doit s'initialiser (en particulier au démarrage du service). Le service n'emettra aucun événement perceptif tant que l'initialisation n'aura pas eu lieu. Le second événement est la notification (par le client applicatif) que l'initialisation peut avoir lieu. Il doit bien entendu obtenir la garantie que les utilisateurs ont « ôté leurs mains » de la surface interactive, par exemple en affichant un message approprié. Le troisième événement est émis lorsque le service est initialisé et prêt à fonctionner.

Afin de maintenir un seuil fonctionnel bas, ce mécanisme est optionnel : si aucun client n'est connecté au canal *reset*, le service s'initialise de lui-même — supposant ainsi que les conditions d'initialisation sont réunies.

gml.tracker.widgets

agent		*qualité de service*		*image & cadrage*		*eclairage*	
doigt		#agents	0 à 1	définition		structure	
propriété d'intérêt		latence		résolution	1 à 2 mm	variation	
aspect		précision	10 mm	couleur	infrared	*confusion*	
géométrie		autonomie		mobilité	☒	complexité	☐
identité	☐☐☐☐			contexte	☒	variation	☐

Certains de besoins concernant les interfaces tactiles ne peuvent être satisfaits par `gml.tracker.finger` car leur environnement d'usage est trop peu contraint : dans cet environnement, le service fournirait une qualité de service trop faible. Le service gml.tracker.widget a pour objectif de satisfaire ces besoins en créant des « widgets virtuels » sensibles aux occlusions par un doigt. À l'initialisation, Laurence désigne au service des zones à observer (surface des widgets). Elle est notifiée lorsque Caroline place son doigt sur la zone en question.

gml.tracker.widget permet le recouvrement de S14, et un recouvrement partiel de S7, S9, S10 (avec une qualité de service moindre).

Sur le canal par défaut *tracking-events* ce service accepte des événements de création et suppression de widgets, et émet en retour des événements de confirmation :

- `<CreateQuery>` (*id*, *x*, *y*, *ox*, *oy*, *sx*, *sy*)
- `<Create>` (*id*, *x*, *y*, *ox*, *oy*, *sx*, *sy*)
- `<DeleteQuery>` (*id*)
- `<Delete>` (*id*)

Ils permettent à Laurence de définir ou supprimer une ou plusieurs zones active, identifiées par leur identifiant *id*. Les événements de confirmation sont émis lorsque les widgets virtuels sont prêts à fonctionner. Une fois un ou plusieurs widgets définis, le service émet les mêmes événements de suivi que `gml.tracker.finger` :

- `<Appear>` (*id*, *x*, *y*)
- `<Update>` (*id*, *x*, *y*)
- `<Disappear>` (*id*)
- `<Dwell>` (*id*, *x*, *y*)

Ici *id* désigne l'identifiant de widget, pas celui d'un agent interactif.

`gml.tracker.widgets` fournit donc un service proche du service précédent, avec trois différences principales : sa précision est beaucoup plus faible (10 mm au lieu de 1 mm), il ne produit pas d'information d'orientation, et l'agent interactif observé est un **widget**. Il ne permet pas de distinguer plusieurs doigts, mais permet l'interaction avec plusieurs doigt puisque plusieurs widgets peuvent être déclenchés simultanément.

5.3.2.3 Interfaces gestuelles

Les interfaces gestuelles rencontrée diffèrent fonctionnellement par le type d'agent interactif étudié : la main, le visage, ou le corps entier. Nous fournissons un service pour chacun des types.

gml.tracker.hand

agent		*qualité de service*		*image & cadrage*		*eclairage*	
main		#agents	0..8	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	5 mm	couleur	infrared	*confusion*	
géométrie		autonomie		mobilité		complexité	
identité				contexte		variation	

Le service `gml.tracker.hand` de suivi de main permet le recouvrement de S8 et S11. Il produit les événements de suivi usuels, associés à des informations géométriques, sur son canal par défaut :

- `<Appear>` (*id*, *x*, *y*, *ox*, *oy*, *sx*, *sy*, *fingers*)
- `<Disappear>` (*id*)
- `<Update>` (*id*, *x*, *y*, *ox*, *oy*, *sx*, *sy*, *fingers*)

fingers est le nombre de doigts visibles sur la main *id*. L'ensemble des informations fournies par les événements permet de déterminer un posture de chaque main : orientation, dimension, et nombre de doigts visible. Comme les techniques existantes sont variées et que les postures à identifier ne sont connues que du client, le service ne fournit par directement ce résultat : ce calcul est laissé au client.

gml.tracker.face

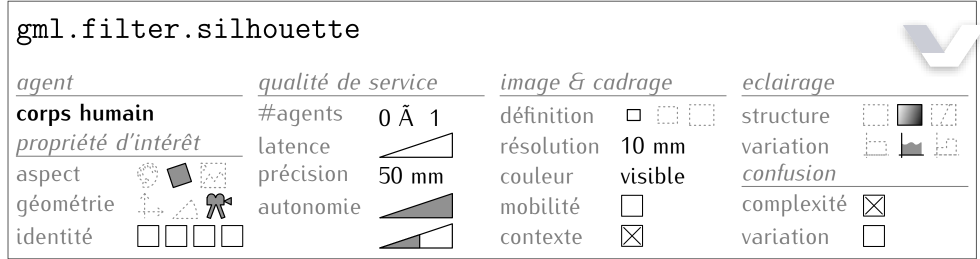
agent		qualité de service		image & cadrage		eclairage	
visage		#agents	0 à 1	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	5 mm	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☐☐☐☐			contexte	☐	variation	☐

Le service gml.tracker.head a pour objectif de suivre le visage et la tête d'un utilisateur. Les résultats du suivi sont des positions en coordonnées caméra. Ce service permet le recouvrement fonctionnel de S16. Il est utilisable en condition « studio », c'est-à-dire lorsque l'utilisateur fait face à la caméra posée à longueur de bras. Cette situation est typique de l'utilisation de *webcams* intégrées à l'écran d'un ordinateur portable ou posées sur l'écran. Ce service est également adéquat pour S15. Son usage correspond alors à l'emploi de la caméra intégrée aux *smartphones* et PDAs récents.

Sur son canal par défaut, il produit les événements suivants :

- <Appear> (x, y)
- <Disappear> $(\varnothing)$
- <Update> (x, y)
- <Blink> (x, y)

L'événement <Blink> est émis lorsque l'utilisateur cligne les deux yeux. Ce service permet de reproduire les démonstrations de la *Fenêtre Perceptuelle* et de *Peephole Displays*.



gml.filter.silhouette

agent		qualité de service		image & cadrage		eclairage	
corps humain		#agents	0 Ã 1	définition		structure	
propriété d'intérêt		latence		résolution	10 mm	variation	
aspect		précision	50 mm	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☐☐☐☐			contexte	☒	variation	☐

Le suivi de la silhouette gml.filter.silhouette permet en particulier de reproduire *VideoPlace* [Krueger et al., 1985], en recouvrant le besoin S17.

- <Appear> (*image*)
- <Update> (*image*)
- <Disappear> $(\varnothing)$

Aux événement de sortie est associée *image* : une représentation de la silhouette. Il s'agit d'une image binaire, éventuellement sous-échantillonnée, contenant une seule composante connexe. Elle représente la silhouette de manière compacte afin de permettre une latence utilisable.

5.3.2.4 Interfaces tangibles

Les besoins exprimés au chapitre 3 pour les interfaces tangibles peuvent se résumer ainsi : le but est la localisation (éventuellement suivi) d'objets spécifiques à une tâche. Ces objets sont connus, et donc modélisables, *a priori*. Ils sont couplés à une surface interactive en utilisant le système de coordonnées de l'interface graphique. Les objets sont par exemple les Post-It's du *Designer's Outpost*, les jetons colorés de la *Table Magique*, les phycons de *Caretta*, ou les *Navigational Blocks*.

Nous proposons de fournir trois services qui se superposent partiellement pour recouvrir cet ensemble de besoins : un suivi d'objets de couleur et de forme connues, un suivi de code-barres bidimensionnels, et un identificateur d'objet.

gml.tracker.color							
agent		*qualité de service*		*image & cadrage*		*eclairage*	
objet coloré		#agents	0 à 20	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☒☐☐☐			contexte	☒	variation	☒

Le service `gml.tracker.color` permet de suivre les jetons et les post-its ; il permet donc de recouvrir de **S4** et **S4'**. Comme se service utilise la perception de l'**aspect** des agents interactif, et pas seulement leurs propriétés, il n'est pas possible de définir une fois pour toutes un modèle des objets à suivre. Il faut donc permettre à Laurence de spécifier les objets à suivre :

- ⇦ `<ModelQuery>` (*id*, x, y, r_0, r_1)
- ⇨ `<Model>` (∅)

À l'initialisation, Caroline place un exemplaire des objets à suivre sur la surface interactive ; Laurence spécifie deux zones circulaires (centrée en x, y), l'une de rayon r_0 inclue dans l'objet, l'autre de rayon r_1 contenant l'objet. En retour, le service construit un modèle de l'objet. Il peut ensuite commencer à emettre les événements perceptifs :

- ⇨ `<Appear>` (*id*, x, y)
- ⇨ `<Update>` (*id*, x, y)
- ⇨ `<Disappear>` (*id*)

Enfin, pour transmettre l'aspect des agents à Laurence, il est préférable de ne pas surcharger les événements de suivi. L'image des agents est utilisée uniquement lors d'interactions faiblement couplées, et l'ajout de *image* aux événements ci-dessus augmenterait déraisonnablement la latence du service. Un couple requête-réponse est donc utilisé :

- ⇦ `<ImageQuery>` (*id*)
- ⇨ `<Image>` (*id*, *image*)

Il est possible de généraliser cette API en utilisant plusieurs modèles, par exemple pour utiliser des jetons de couleurs ou de formes différentes sur la *Table Magique*. Il serait alors nécessaire de munir chaque modèle d'un identifiant, attaché à `<Model>`, `<Appear>`, et `<Update>`.

gml.identifier.object							
agent		*qualité de service*		*image & cadrage*		*eclairage*	
objet		#agents	1	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	100%	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☐☐☒			contexte	☐	variation	☐

Le service `gml.identifier.object` a le rôle de reconnaître certains objets dans une image grâce à leur apparence. En combinant `gml.tracker.color` avec le service gml.identifier.object, Laurence obtient un recouvrement de **S12**. Le service traite les événements suivants :

- ⇦ `<IdQuery>` (*qid*, *id*, *image*)
- ⇨ `<Id>` (*qid*, *id*)

Les événements de type `IdQuery` sont utilisés par Laurence aussi bien pour définir l'apparence des objets d'intérêt que pour bénéficier du service d'identification proposé : si une identité d'objet est fournie ($id \neq 0$), le service associe l'image à l'identité fournie. Si $id = 0$, le service détermine l'identité correspondante à l'image fournie.

Afin de permettre des requêtes concurrentes, un identifiant de la requête est utilisé : *qid*.

`gml.tracker.tag`

agent		*qualité de service*		*image & cadrage*		*eclairage*	
tag		#agents	0 à 1000	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm / 10°	couleur	infrared	*confusion*	
géométrie		autonomie		mobilité		complexité	
identité				contexte		variation	

Lorsque le nombre d'identités distinctes pour les agents interactifs augmente, ou que les agents d'identité différente se ressemblent, il n'existe pas de méthode de vision permettant d'identifier l'agent observé avec une robustesse raisonnable. Une solution courante consiste à étiqueter physiquement les agents avec des codes-barre bidimensionnels (ou *tags*) comme ceux de l'*ARToolkit* [Kato et al., 2000] ou du système *ARTag* [Fiala, 2005].

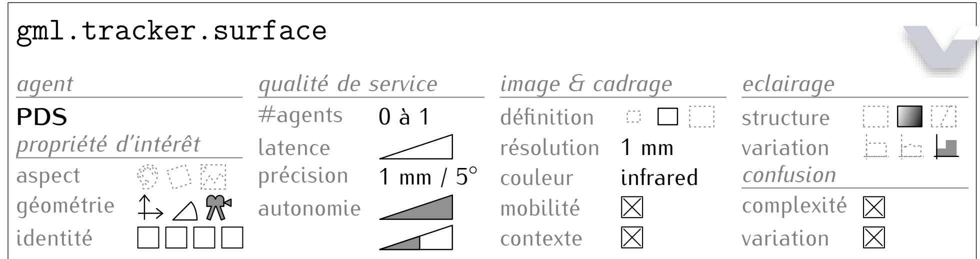
Le service `gml.tracker.tag` permet le recouvrement de **S1**, **S2** et **S3**, et **S12**. Les événements émis par le service sont les événements usuels à ceci près que les identités sont absolues (i.e. non définies par le service) :

- `<Appear>` (*id*, *x*, *y*, *ox*, *oy*, *s*)
- `<Update>` (*id*, *x*, *y*, *ox*, *oy*, *s*)
- `<Disappear>` (*id*)

Le service fournit également l'orientation des *tags*, ainsi que leur échelle (en millimètres projetés sur la surface interactive).

5.3.2.5 Interfaces mobiles

Une partie des besoins perceptuels pour les interfaces mobiles a été couvert par `gml.tracker.face`. Lorsque les déplacements de la surface interactive mobile observée sont absolus, un nouveau service doit être fourni.



Le service `gml.tracker.surface` permet de reconstruire le prototype *PDS* [Borkowski et al., 2003]. Il permet donc le recouvrement de **S15'**. Le service produit en sortie la matrice de transormation projective entre les coordonnées utilisées par le projecteur video et les coordonnées normalisées sur la surface portable (les coordonnées qu'utilise Laurence lorsqu'elle définit l'interface graphique portable).

- `<Appear>` (*matrix*)
- `<Update>` (*matrix*)
- `<Disappear>` (∅)

Par exemple, si Laurence utilise OpenGL pour afficher l'interface sur la surface portable, il lui suffit d'empiler la matrice fournie par le service avant de commencer chaque procédure affichage.

5.3.2.6 Interfaces implicites

Notre état de l'art indique que les surfaces implicites nécessitent la détection et la localisation grossière de personnes relativement au capteur. De plus, savoir si la personne fait face au capteur est une valeur ajoutée.

gml.tracker.skin							
agent		*qualité de service*		*image & cadrage*		*eclairage*	
visage		#agents	0 à 10	définition	☐	structure	
propriété d'intérêt		latence		résolution	10 mm	variation	
aspect		précision	1 m	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☒	complexité	☒
identité	☒☐☐☐			contexte	☒	variation	☒

En detectant et suivant les régions de la vue de la caméra identifiées comme de la peau, le service `gml.tracker.skin` permet de recouvrir **S13**. Il s'agit d'un suivi classique :

- `<Appear>` (x, y, *id*, *facing*, r)
- `<Update>` (x, y, *id*, *facing*, r)
- `<Disappear>` (*id*)

Deux informations supplémentaires sont ajoutées aux événements : *facing* est un booléen indiquant si la zone détectée est un visage faisant face au capteur, et r est le rayon de la zone détectée (en pixels).

Comme il n'est pas possible de reconnaître la peau sans connaître les caractéristiques de l'éclairage et du capteur [Letessier, 2003], le service fournir à Laurence un moyen d'initialiser le système :

- `<SetupQuery>` (∅)
- `<Setup>` (∅)

La requête `<SetupQuery>` permet de déclencher l'initialisation du système. Caroline doit alors faire en sorte que des utilisateur fassent face au capteur. Lorsque le système a perçu suffisement de visages et peut fonctionner, il envoie `<Setup>` en retour.

5.3.3 Services de capture

Nous avons déjà présenté deux services qui produisent des images : `gml.filter.silhouette` et `gml.tracker.color`. Ces images sont des les résultats de traitement de perception. D'autres services sont nécessaires en amont pour permettre la perception : les sources d'images, qui représentent une abstraction du matériel d'acquisition vidéo ou d'un flux video enregistré, et le *scanner*, qui permet de numériser à haute résolution l'aspect d'une surface interactive ou d'un objet.

5.3.3.1 Sources d'images : `gml.grabber.*`

Par définition, tous les services de vision artificiel utilisent en entrée un flux video. Nous pensons que, d'un point de vue centré utilisateur, il convient d'isoler la construction de ces flux video dans des services plutôt que de les intégrer dans le service perceptif qui les utilise. Les justifications sont tirées des requis structurels présentés au chapitre 4 :

réutilisation et partage Une source video peut être utile à plusieurs services. C'est ce que nous avons montré sur la figure 4.4 page 83. En utilisant des services, il est trivial de partager l'accès à la ressource : il suffit que plusieurs clients établissent une connexion au service caméra.

prototypage Stanislas et Patrick ont l'habitude de tester initialement leurs algorithmes perceptifs sur des vidéos enregistrées, plutôt que des vidéo *live*. Ceci permet de tester des cas particuliers, faire des mesures de performances, et obtenir des résultats reproductibles. En utilisant un service video *offline* et un service *live*, il suffit de connecter l'un ou l'autre au service perceptif pour étudier l'une ou l'autre condition.

robustesse L'acquisition video est la couche de « plus bas niveau » d'un système perceptif — la plus proche du matériel. L'isoler permet donc d'éviter que des plantages se propagent jusqu'à faire échouer le service perceptif, et/ou le système interactif dans son ensemble. En outre, en cas de problème, le service video peut être relancé et reconnecté au client dynamiquement.

indécision Un système interactif peut utiliser plusieurs caméras. C'est le cas de la *Table Magique* et du *Designer's Outpost*, par exemple. Seul la personne installant le système peut déterminer quel caméra doit être utilisé pour quel service perceptif : en isolant chaque caméra dans un service, l'utilisateur bénéficie de la flexibilité de l'interconnection dans notre architecture à services.

gmlVision fournit par conséquent deux services de production de flux video.

Le service `gml.grabber.camera` est l'abstraction d'une caméra ou d'un appareil photo réel. Il produit des images en « temps réel, » c'est-à-dire les fournit au client sans les transformer et le plus tôt possible après leur acquisition.

Le service `gml.grabber.offline` est un lecteur d'images. Il fournit les images d'un flux video enregistré, sous forme d'une séquence de fichiers image ou d'un fichier video.

⇨ `<Image>` (*image*)
⇦ `<ImageQuery>` (∅)

Les deux services possèdent deux canaux : *stream* émet toute les images capturées, et *frame* émet des images à la demande. `<Image>` est donc émis par les deux canaux, et `<ImageQuery>` accepté uniquement par le second.

5.3.3.2 Scanner

Dans les besoins **S5** et **S5'** (*Designer's Outpost*, et *Table Magique*) apparaît la nécessité de pouvoir capturer l'aspect d'une portion de la surface interactive avec une résolution élevée. `gml.grabber.surface` remplit cet office :

⇦ `<ImageQuery>` (*vertices*)
⇨ `<Image>` (*image*)

L'utilisateur spécifie une portion de la surface interactive en émettant `<ImageQuery>` associé aux coordonnées des sommets du polygone à capturer (dans les coordonnées de l'interface). En retour, le service produit une image rectifiée. Dans le cas de la *Table Magique*, le service doit pouvoir être configurer pour « nettoyer » l'image, c'est-à-dire segmenter les traits dessinés sur la surface par les utilisateurs (en utilisant par example l'approche décrite dans [Wellner, 1993a])

5.3.4 Services support

Nous présentons dans cette section l'ensemble des services support de *gmlVision*. Il s'agit des services permettant d'améliorer l'**autonomie** et la **robustesse** de l'ensemble des services perceptifs, mais ne fournissant pas directement de service pour l'interaction à Laurence.

5.3.4.1 Calibrage géométrique : la famille `gml.calibrator`

Parmi les services perceptifs présentés, une majorité produit ou reçoit des événements associées à des coordonnées de l'interface graphique projetée, telle que manipulées par Laurence. C'est par l'opération de calibrage géométrique que le système perceptif peut établir la correspondance entre les coordonnées de l'image perçue par la caméra et celles de l'interface.

Dans les systèmes étudiés, et avec le materiel rencontré, les déformations sphériques et autres écarts au modèle sténopé sont négligeables ; cette correspondance est donc modélisée par une **matrice** de transformation projective.

Les trois services de calibrage.

Différents protocoles de calibrage 2D/2D sont mis à disposition selon les besoins d'autonomie et les possiblités du dispositif expérimental. Le premier, `gml.calibrator.assisted`, sera le plus rapide, mais il est assisté : il suppose que Caroline a aligné la vue de la caméra avec la projection (i.e. que l'ordre des points de la mire est le même en coordonnées interface et en coordonnées caméra). Si l'alignement est imparfait, le calibrage échoue.

Le second, `gml.calibrator.automatic`, permet de gérer des situations plus complexes (fortes déformations projectives, ou recouvrement partiel entre vue caméra et affichage). Il est plus autonome pour Caroline (qui n'a pas a aligner la caméra et le projecteur). En contrepartie, il impose à Laurence de déployer ou d'implémenter un service d'affichage `gml.display` (décrit dans la section suivante).

Enfin, le troisième service, `gml.calibrator.manual`, est adaptée aux caméras infrarouge : la caméra ne pouvant percevoir la projectoin, il est impossible d'effectuer un calibrage automatique. C'estalors à Caroline d'établir les correspondances entre des points de la vue caméra et des points de l'affichage.

Description des protocoles.

L'ensemble des protocoles ont des points communs : deux canaux doivent être présents, le canal *video* qui reçoit les images d'un service caméra :

⇦ `<Image>` (*image*)

et le canal *matrix*, qui émet la matrice de transformation des coordonnées caméra aux coordonnées interface, et les dimensions de la région d'intérêt, en coordonnées interface :

⇨ `<Matrix>` (*matrix*, *sx*, *sy*)

Nous présentons les protocoles des trois services de calibrage par ordre chronologique de leur développement.

`gml.calibrator.assisted`

fournit une marnière simple d'effectuer le calibrage. Via le canal par défaut *calibration*, Laurence émet une requête de calibrage ; le service confirme le début de la procédure en émettant `<Calibration>` :

⇦ `<CalibrationQuery>` (∅)
⇨ `<Calibration>` (∅)

Le service demande à Laurence de spécifier les dimensions *sx* et *sy*, en pixels, de la zone de son interface à calibrer.

⇨ `<DefinitionQuery>` (∅)
⇦ `<Definition>` (*sx*, *sy*)

Laurence est responsable de l'affichage de la mire de calibrage, qui doit être constituée de rectangles disposés sur une grille et fortement contrastés avec le fond. Afin d'introduire une contrainte d'orientation sans laquelle le calibrage est indéterminé, le repère de l'interface graphique et celui de l'affichage doivent être grossièrement alignés au moment de l'installation du dispositif. La grille de calibrage est spécifiée par *centers*, la liste des coordonnées des centres des points de la grille, en coordonnées interface :

⇨ `<GridQuery>` (∅)
⇦ `<Grid>` (*centers*)

Le calibrage est donc **assisté** par l'utilisateur final, Caroline, d'où le nom du service. Si la mire est mal alignée, ou si l'un de ses point n'a pu être détecté, le service émet un événement d'erreur ; Laurence peut alors informer Caroline que la caméra doit être positionnée et réglée correctement.

⇨ `<Error>`

Une fois le calibrage terminé, le service donne à Laurence une estimation de l'erreur maximale de calibrage (en pixels interface). Ceci lui permet de recommencer le calibrage si le résultat est trop mauvais.

<Done> (*error*)

`gml.calibrator.automatic`

fournit un calibrage automatique : aucune intervention de Caroline n'est nécessaire. Son API est différente de celle de `gml.calibrator.assisted`. Via le canal par défaut *calibration*, Laurence n'effectue que la requête de calibrage :

<CalibrationQuery> (∅)
<Calibration> (∅)
<Error>
<Done> (*error*)

L'affichage de la mire de calibrage est négociée via le canal *display*. Lors de l'exécution de la procédure, le service demande au client la définition de l'interface graphique (*sx*, *sy*, en pixels) puis d'afficher une mire en émettant plusieurs requêtes d'affichage <PolygonQuery>. Le service perceptif détecte alors les points de la mire dans la scène afin d'établir la matrice de transformation.

<DefinitionQuery> (∅)
<Definition> (*sx*, *sy*)
<PolygonQuery> (*id*, *color*, *vertices*)
<Polygon> (*id*)
<DeleteQuery> (*id*)
<Delete> (*id*)

Le service `gml.display` est capable de communiquer avec le canal *display* de ce calibreur. Il est fournit par *gmlVision* et décrit ci-dessous ; les événements correspondant y sont détaillés.

`gml.calibrator.manual`

Lorsque la caméra observe la scène dans l'infrarouge, il n'est généralement pas possible d'effectuer un calibrage automatique ou assisté. En effet, un projecteur video, un écran cathodique, ou un ecran à cristaux liquides n'émettent de lumière que dans la partie visible du spectre : les spectres d'emissions et de reception ne s'intersectent pas, le calibrage doit donc être manuel.

Ce service fournit les mêmes canaux et la même API que `gml.calibrator.automatic`. Pour permettre la mise en correspondance des points de la mire avec les points perçus, le service est doté d'une interface graphique, qui affiche la vue de la caméra. Via cette interface, Caroline désigne les centres des points de la mire, permettant de déterminer leurs positions perçues.

Remarquons que les points de la mire ne sont pas visibles pour Caroline sur l'interface du service, puisque la vue affichée est celle de la caméra — qui ne perçoit que dans l'infrarouge. La marche à suivre est alors de placer un objet physique sur chacun des points de la mire affichée ; les objets physiques seront, eux, visibles sur l'écran.

5.3.4.2 Affichage : `gml.display`

Pour les différents types de calibrage il est nécessaire d'afficher une mire sur la surface interactive. Mais c'est le client applicatif qui contrôle l'interface graphique présente sur la surface interactive. Il est d'ailleurs possible que l'interface et le service perceptif ne soient pas colocalisés (i.e. ne s'éxécutent pas sur la même machine).

Afin de permettre l'affichage de la mire, *gmlVision* doit donc fournir un service `gml.display`, qui peut être embarqué dans l'application cliente ou exécutée indépendemment (quoique sur la même machine).

Les événements de dessin sont inspirés des commandes du canevas GML [Bérard, 2006], une boîte à outils de construction d'interface graphique que nous avons contribué à concevoir et développer.

Sur son canal par défaut, `gml.display` accepte plusieurs types de requêtes. La première permet à un client de déterminer la définition (dimensions en pixels) de la surface interactive :

`<DefinitionQuery>` (∅)
`<Definition>` (*sx*, *sy*)

La seconde permet le dessin de polygones quelconques :

`<PolygonQuery>` (*id*, *color*, *vertices*)
`<Polygon>` (*id*)

La dernière permet de supprimer un élément graphique :

`<DeleteQuery>` (*id*)
`<Delete>` (*id*)

L'identifiant particulier *id* = *0*, n'est jamais utilisé, sauf en paramètre de `<DeleteQuery>` ; il permet alors d'effacer tous les éléments graphiques.

5.3.4.3 Abstraction d'une surface interactive : `gml.surface`

Malgré les efforts fournis pour atteindre une boîte à outils au seuil faible et à l'autonomie élevée, une surface interactive peut devenir complexe à mettre en place dès que ses différentes composantes logicielles ne sont plus colocalisées. Cinq services doivent être connectés : la caméra, l'affichage, le service de calibrage, le service perceptif, et le client applicatif.

Dans le cas le plus simple, Caroline utilise une seule machine à laquelle est connecté une unique caméra et un unique projecteur. La connection entre les cinq services est alors triviale. Mais d'autres situations sont probables : plusieurs caméras, plusieurs affichages connectés à une seul machine, ou bien deux machines distinctes exploitant respectivement la caméra et l'affichage.

Connecter ces services entre eux, et maintenir ces connections en cas redémarrage d'un des services, est une tâche pénible pour Laurence : elle élève le seuil fonctionnel. D'autre part, seule Caroline est à même de spécifier quels services doivent être connectés ensemble, puisque la répartition des services n'est connue qu'au déploiement : il existe donc également un problème d'autonomie et de seuil pour Caroline.

Il nous paraît donc pertinent que *gmlVision* fournisse un service qui agit comme la « glu » entre les composantes. Le service `gml.surface` est chargé de définir et maintenir établies les connexions entre les canaux des différents services. Il peut, typiquement, être embarqué par Laurence dans l'application cliente. Il présente une interface graphique permettant à Caroline de spécifier les cinq composantes à connecter — parmi des services découverts, ou instanciés localement par Laurence. Par contre, il ne fournit aucune API orientée événement (ce n'est pas un service perceptif). Il agit en contrôlant les autres services *via* leur canal de contrôle.

Ce service est une **matérialisation du couplage** entre la caméra, le service perceptif, et le projecteur ; c'est-à-dire, une matérialisation de la surface interactive.

5.3.4.4 Calibrage de l'éclairage : `gml.filter.lighting`

Nous avons évoqué au chapitre 3 la difficulté de réaliser des systèmes perceptifs **robustes**. Pour parvenir à une robustesse acceptable (c'est à dire offrant un système utilisable), il faut fournir à Patrick des outils pour mesurer l'environnement. Ceci peut lui permettre, par exemple, de stopper le service si la mesure montre que le système ne fonctionne plus dans un environnement supporté.

La plupart des services perceptifs ne peuvent pas fonctionner lorsque l'image est surexposée ou sous-exposée. Par exemple il est fréquent de rencontrer des cas d'usage de surfaces augmentées où la caméra et le projecteur video sont quasi-coaxiaux, ce qui produit un phénomène de *bright spot* (zone lumineuse surexposée) sur la surface.

Il est necessaire de rendre ces phénomènes observable pour Laurence. Elle peut choisir de construire son interface graphique en « évitant » les zones insensibles, ou les représenter, afin de rendre la situation également observable pour Caroline.

Connecté à un service `gml.grabber.camera`, le service `gml.filter.lighting` est chargé de calculer des métriques sur l'éclairage, de détecter des changement d'éclairage, d'évaluer le bruit moyen, de produire une carte de bruit pour d'autres services. Il peut également permettre de configurer la caméra pour qu'elle fournisse la « meilleure » image pour la vision artificielle.

Du point de vue le Laurence, l'API d'intérêt est celle qui permet de connaître les régions de la surface qui seront rendues inertes car la perception ne peut y avoir lieu, en particulier les régions surexposées ou sous-exposées.

Le canal *video* accepte les images d'un `gml.grabber.*` :

⇠ `<Image>` (*image*)

Le canal de sortie *badmap* produit une carte des lieux de la surface interactive où l'image est trop mauvaise (trop bruitée, surexposée, ou sous-exposée par exemple) pour être utilisée par les services perceptifs. Cette carte est utilisable par Laurence pour produire un *feedback* pour Caroline — c'est-à-dire rendre observables les problèmes éventuels.

⇢ `<Image>` (*image*)

Le flux video de sortie est une séquence d'images binaires, sous-échantillonnées dans l'espace et le temps, afin de maintenir une latence faible.

5.4 Conclusion

Dans ce chapitre, nous avons présenté notre approche pour concevoir *gmlVision*, puis nous avons détaillé son architecture et son contenu fonctionnel. Du point de vue de l'utilisateur, une approche spécifique (*ad hoc*) pour la conception de la bibliothèque au plus haut niveau est préférable à une approche générique.

D'après un ensemble de critères centrés utilisateurs, nous proposons une architecture externe orientée services à Laurence. Chaque service est une « boîte noire » pourvue de « prises » permettant de la connecter à d'autres services ou à l'application interactive et d'échanger des flots d'événements perceptifs ou de contrôle.

L'interface interne est axée sur la minimisation du seuil fonctionnel d'une part, et l'implémentation d'un framework à services d'autre part. Ceci permet à Patrick et Stanislas un apprentissage facile de *gmlVision*, et le développement rapide de services en leur fournissant un chemin de moindre résistance. Pour les développeurs de *gmlVision*, une architecture à composants légers permet de satisfaire aux contraintes de la perception artificielle. Enfin, des *patterns* de développement sont fournis pour guider Patrick et Stanislas vers un développement efficace.

Nous proposons par ailleurs que, pour atteindre le recouvrement fonctionnel des besoins, une démarche empirique et itérative est la plus adaptée. Cette démarche nous

permet de satisfaire l'ensemble des besoins expimés dans le chapitre 3. La démarche doit également être utilisée par Patrick pour étendre la bibliothèque pour couvrir de nouveaux besoins. L'ensemble des services fournis se structure en 3 parties : les services perceptifs, les services d'abstraction du matériel, et les services « support » permettant la configuration et la régulation automatique d'un système interactif.

Nottons que la méthode et l'architecture proposées semblent capables de perrénité. Dans un contexte où l'environnement peut contenir de multiples capteurs et de multiples lieux et tâches d'interaction, une architecture découplée est dynamique est adaptée.

Le niveau de granularité des services et le niveau de complexité des APIs choisies nous paraissent également adaptés aux évolutions des matériels. Il nous paraît réaliste d'exposer la fonctionnalité d'une *smart camera* [Desurmont et al., 2005] sous la forme d'un service *gmlVision*, voire d'intégrer des services de perceptions aux prochaines génération de *smart cameras*. Il est également possible d'imaginer des projecteurs intelligents de la même manière.

À présent que l'ensemble de l'interface de *gmlVision* est définie, nous devons présenter les technologies logicielles permettant de l'implémenter. Nous pourrons ensuite évaluer notre proposition *via* l'évaluation de cette implémentation.

6 Mise en œuvre de *gmlVision*

> *« Programming, like music, blends esthetics and technology. The high-level plan, the middle-level concepts, and the low-level details must be correct and in harmony with each other. Discordant data structures or missed notations are jarring. »*
>
> *[Shneiderman, 1986]*

Le thème central de ce chapitre est le choix : celui de technologies, de techniques, et d'algorithmes qui permettent de répondre aux spécifications énoncées au chapitre précédent. Ici encore, ces choix sont guidés par les besoins des utilisateurs de la boîte à outils.

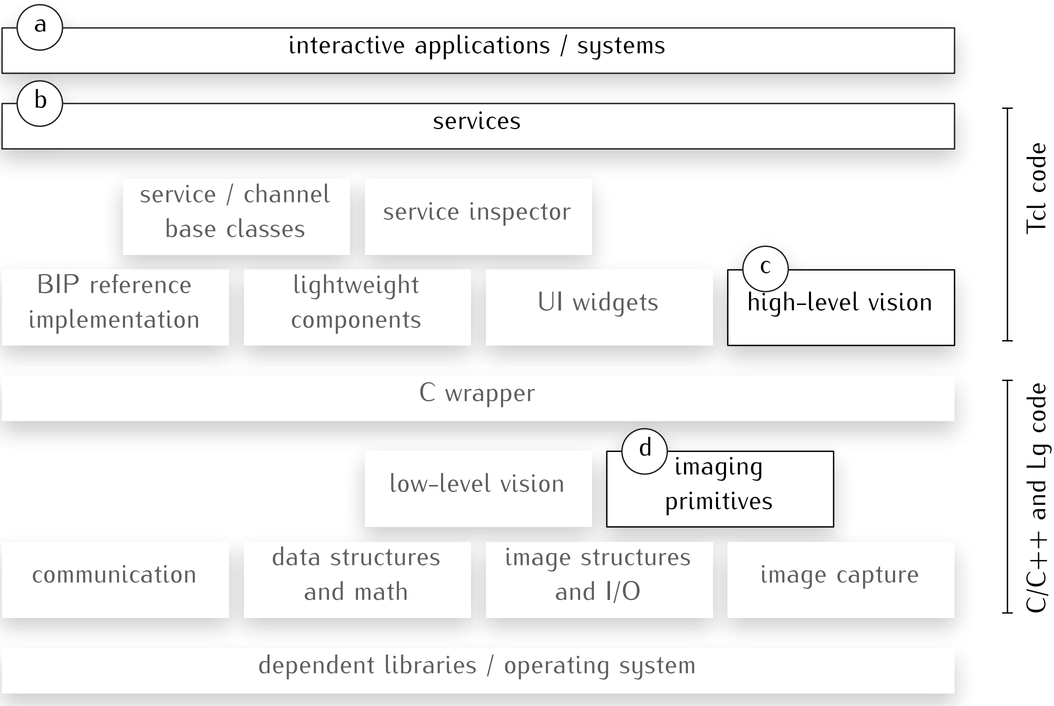


Figure 6.1 . Schéma-bloc de la structure de la bibliothèque *gmlVision*.

Les bloc sans bordure sont ceux implémentés par les développeurs de *gmlVision*. Les blocs délimités en noir sont ceux implémentés par Laurence (*a*), Patrick (*b*), et Stanislas (*c* et *d*). Dans le cadre de cette thèse, nous avons adopté chacuns des trois rôles pour implémenter tout ou partie de ces blocs.

La structure générale de *gmlVision* est résumée sur la figure 6.1 page précédente. Chaque bloc représente un ensemble de modules indépendant des autres. En d'autres termes l'API d'un bloc peut être utilisée sans connaissance de l'API des autres blocs. Le schéma est structuré en couches, du plus bas au plus haut niveau d'abstraction. Un bloc placé au-dessus d'un autre en est fonctionnellement dépendant.

Il convient de présenter cette structure du point de vue des différents utilisateurs. Les blocs en gras sont ceux développés par Patrick et Stanislas. En se basant sur la figure 3.1 page 36, précisons leurs rôles vis-à-vis de la bibliothèque :

Laurence utilise les services empaquetés, c'est-à-dire des applications exécutables qui instancient un service, et dans le cas générale offrent une interface graphique permettant de le configurer.

Patrick construit des services et les empaquète. À cet effet, il utilise le squelette de service fourni (classe mère `Service`), les canaux de communication standard (classe `ServiceChannel` et ses classes dérivées), et les composants fournissant des fonctions de vision par ordinateur (implémentés par Sanislas).

Stanislas crée des primitives de traitement d'image (au niveau C). Au niveau Tcl, il manipule ces primitives, les structures de données et les outils mathématiques fournis pour construire des composants vision.

Dans la première section (6.1), nous présentons l'architecture concrète de la boîte à outils, c'est-à-dire le support structurel au développement de services et de composants. Nous décrivons les deux *middleware* du point de vue des utilisateurs et donnons quelques détails sur leur implémentation.

Nous poursuivons (6.2 page 127) en décrivant le contenu fonctionnel actuellement disponible pour Laurence : les services perceptifs et services support actuellement implémentés, qui correspondent à la conception du chapitre précédent. Nous détaillons leur fonctionnement en insistant sur les choix techniques permettant de remplir le contrat.

6.1 Architecture concrète de *gmlVision*

Nous présentons dans cette section les choix de réalisation de l'architecture de *gmlVision* dont la conception est présentée dans la section 5.2. Nous décrivons ici les API internes et externes du middleware à services utilisé dans *gmlVision* par Laurence et Patrick, puis du middleware à composants utilisé par Patrick et Stanislas.

6.1.1 *gmlBIP* : un middleware pour les services perceptifs

Notre conception retient une architecture à service pour l'interface externe de *gmlVision*. Cette architecture doit supporter la découverte et détection de services, un ensemble de canaux de communications, une communication événementielle asynchrone, et une interface d'inspection et de contrôle.

Le protocole BIP (*Basic Interconnection Protocol*) a été conçu pour satisfaire ces requis [Borkowski et Letessier, 2006]. Sa spécification est reproduite en annexe A. Il fait l'objet de deux implémentations : *gmlBIP* est une implémentation homogène à *gmlVision* mais qui peut être utilisée de façon indépendante. *O3MiSCID* (*Open object-oriented middleware for service connection, introspection and discovery*) [Emonet et al., 2006] est une implémentation plus performante en C++ et Java développée dans l'équipe PRIMA.

Nous justifions en premier lieu la conception de BIP en montrant en quoi les middlewares existants ne satisfont pas nos requis. Nous présentons ensuite l'API interne et externe de *gmlBIP*. Nous concluons sur un ensemble de considérations sur la réalisation de gmlBIP et remarques sur les extensions apportées à BIP dans le cadre de *gmlVision*.

6.1.1.1 Middlewares existants et choix technologiques

Une taxonomie des middlewares est proposée par [Ritter, 1998] en fonction des mécanismes d'échange d'information qu'ils fournissent :

- Appel de procédure distant (RPC, *Remote Procedure Call*). Les clients demandent au service l'exécution d'une procédure. La transaction peut être synchrone ou non. Le service n'a pas d'état.
- Middleware orienté messages (MOM, *Message Oriented Middleware*). Des messages sont envoyés au client. Ils sont stockés dans une file d'attente jusqu'à être traités (par exemple par ordre d'arrivée ou de priorité). Pendant ce temps, le client continue d'autres traitements.
- Courtage de requêtes objet (ORB, *Object Request Broker*). Chaque objet du système peut invoquer des méthodes d'autres objets distants, de manière synchrone ou non. L'ORB véhicule les requêtes. Ceci permet la réalisation de systèmes à objets distribués.

Un MOM semble être le choix le plus pertinent pour pour l'interface d'un service perceptif car c'est le plus proche du modèle de programmation par événements. Les messages échangés seront alors des événements perceptuels ou des requêtes et réponses pour l'invocation distante de méthode.

Les middlewares orientés service existantssont du type ORB. La communauté industrielle les nomme ESB (*Enterprise Service Bus*). C'est par exemple le cas de *Windows Communications Foundation*, anciennement *Indigo* [Microsoft, 2006], ou Apache ServiceMix [LogicBlaze, 2006]. Il est imaginable au prix d'un effort de développement important d'implémenter le modèle MOM au-dessus de tels middlewares. C'est la piste choisie dans le cadre de « SOA 2.0 » par l'ESB libre Mule [Codehaus, 2007] qui implémente une architecture dirigée par les événements (EDA, *Event Driven Architecture*). Cependant, tous ces ESB souffrent également, du point de vue des besoins que nous avons identifiés, d'autres défauts. Nous résumons les plus importants :

- ils utilisent des protocoles de communication uniquement transactionnels (modèles ORB ou RPC), ou bien asynchrones mais pas événementiels (CORBA, SOAP) et sans persistence de connection entre client et service (SOAP/HTTP). Les utiliser pour le modèle MOM est donc peu performant : il est necessaire d'établir une nouvelle connexion à chaque message.
- ils sont indissociables de *frameworks* lourds qui imposent un langage de programmation ou une plate-forme. Les critiques formulées auparavant à l'égard de ce type d'architecture restent valides. Par contraste, nous nous proposons de ne pas imposer d'environnement logiciel particulier.
- l'interopération est possible avec tout couple langage/plate-forme, mais le chemin de moindre résistance n'existe que pour un seul (généralement Java).
- le déploiement est coûteux : de nombreuses dépendances logicielles doivent être satisfaites. Au minimum, une machine virtuelle et son environnement, par exemple, .NET, Mono, ou Java.
- l'autonomie est faible : soit les services et les applications sont connectés *a priori* par le concepteur, soit le système utilise un annuaire de services (*service registry*) pour la découverte, qui doit lui aussi être déployé.

Par contraste, nous voulons que le middleware utilisé soit performant, facilement ré-implémentable dans un nouvel environnement, facile à déployer, et agnostique en termes de plate-forme et de langage de programmation. Voici, par ordre de priorité, les requis sous-tendant la conception de `BIP` :

- le protocole doit contenir un sous-ensemble minimal suffisant pour que les services soient utilisables, tout en permettant l'interopération avec des programmes simples comme `nc` ou `telnet` pour faciliter le diagnostic des problèmes ;
- aucune postulat n'est émis sur le système l'exploitation (OS), le langage de programmation, ou la distance (même processus, même machine, ou même sous-réseau) des *peers* ;
- les *peers* communicant en utilisant `BIP` sont des services « boîte noire » isolés ;

- les services sont nommés de manière lisible par un utilisateur, car l'interconnection sera généralement effectuée par des humains ;
- les services doivent pouvoir être découverts dynamiquement et adressés par leur nom, afin d'éviter les configuration d'adresses réseau et de ports par un utilisateur, sans toutefois utiliser un annuaire dont la mise en place serait trop lourde ;
- de multiples mécanismes de transport de l'information doivent être disponibles afin de satisfaire les besoins de latence et de fiabilité disparates ;

6.1.1.2 Description générale de `BIP`

TCP/IP est de nos jours supporté dans tous les langages et sur toutes les plate-formes matérielles pour permettre la communication entre processus (éventuellement distants). L'objectif de `BIP` est de permettre l'interopération avec TCP/IP seul. Si la plate-forme logicielle permet plus de fonctionnalités d'interopération (UDP, DNS-SD par exemple) et si elles sont exploitées par l'implémentation de `BIP`, le plafond fonctionnel sera plus haut et la qualité de service meilleure. Par contre ces fonctionnalités doivent rester optionnelles : nous nous autorisons des **extensions** au protocole à condition de ne pas élever le seuil.

Dans la taxonomie de la RFC 3117 [Rose, 2001], le protocole `BIP` peut être décrit par les critères suivants :

- encapsulation (comment sont délimités les messages) : comptage des octets par message ;
- encodage (comment les messages sont représentés) : utilisant MIME, par défaut `text/plain`, c'est-à-dire texte ASCII (sauf le canal de contrôle utilise `text/xml` par défaut)
- rapport d'erreur (comment les erreurs sont décrites) : non spécifié, laissé aux couches supérieures au protocole ;
- asynchronisme (comment les échanges indépendants sont gérés) : canaux de communication séparés, possibilité de multiplexer des échanges sur le même canal ;
- authentification et sécurité (comment les *peers* sont identifiés, et comment leurs échanges sont protégés) : minimale, chaque *peer* indique son identité.

Dans son utilisation la plus simple `BIP` spécifie simplement l'établissement d'une connection logique et une encapsulation de messages sur une connection TCP. L'encapsulation consiste en un en-tête textuel de taille fixe ; par exemple :

```
BIP/1.0 A47F64A1 0000001A 00001D3
```

signifie que la version du protocole utilisée est `1.0`, le peer émettant le message a pour identifiant `A47F64A1`, le message est le 26ème envoyé (`1A` en hexadécimal), et sa longueur est de 267 octets (`1D3`). L'établissement de la connection logique s'effectue par le biais du premier message échangé (numéroté 00000000, et appelé *message magique*). Il peut **optionnellement** être utilisé par les deux peers pour des négotiations, par exemple sur le format des informations échangées ou les modes de transport.

Lorsque DNS-SD [Cheshire et Krochmal, 2006] est disponible il devient possible de publier et découvrir des services de manière décentralisée (i.e. sans annuaire). `BIP` spécifie des conventions sur la publication des services : un type de service (`_bip._tcp`), des attributs descriptifs (`class` pour la classe de service, `id` pour l'identifiant unique du service, etc.) et des attributs fonctionnels (désignant les différents canaux de communication).

Lorsque la communication est possible en utilisant un autre mode de transport (UDP par exemple), `BIP` spécifie comment utiliser le message magique pour permettre aux peers d'utiliser ce mode de transport (BIP sur UDP). Ceci permet d'utiliser des transports plus performants que TCP en termes de latence et de débit, en particulier lorsque les deux peers s'exécutent sur la même machine.

Enfin, un peer communicant avec le canal de contrôle d'un service peut inspecter et modifier ses paramêtres, et inspecter et connecter ses différents canaux. BIP spécifie

un schéma XML décrivant le format des requêtes et des réponses sur ce canal (c.f annexe A).

La structure de BIP est résumée par la figure 6.2. Plus de détails techniques sur le protocole sont disponibles dans l'annexe A page 177.

6.1.1.3 Accès aux services

Pour interagir avec un service, au minimum quatre primitives doivent être implémentées : connection à un canal, réception d'événements, émission, et déconnection. Ce sont ces quatres procédures qui doivent pouvoir être implémentées simplement dans tout langage. Voici des prototypes satisfaisants pour ces procédures :

Connect : *host, port* → *handle*
Établit une connection avec un canal présent sur la machine *host* et le port TCP *port*. La connection est matérialisée par *handle*.

Bind : *handle, event, callback* → ∅
Si *event* est `receive`, passe chaque message reçu à la procédure *callback* ; si *event* est `connect` (respectivement `disconnect`), appelle *callback* lorsque la connection est établie (resp. rompue).

Send : *handle, data* → ∅
Envoie le message *data* au service connecté.

Disconnect : *handle* → ∅
Détruit la connection au service.

Pour utiliser en plus les fonctionnalités de découverte de service, une implémentation de BIP doit permettre d'obtenir la liste des instances de service d'une classe donnée, et de permettre la connection d'après un nom de service et de canal :

ServicesByClass : *class* → *service name list*
Renvoie la liste des noms de service d'une classe donnée présents sur la machine hôte et le réseau local.

Connect : *class, [channel name]* → *handle*
Établit une connection avec le canal *channel name* du premier service rencontré de la classe spécifiée. Si le canal n'est pas spécifié, la connection est établie avec le canal par défaut.

Connect : *service name, [channel name]* → *handle*
Établit une connection avec le canal *channel name* du service unique nommé *service name*. Si le canal n'est pas spécifié, la connection est établie avec le canal par défaut.

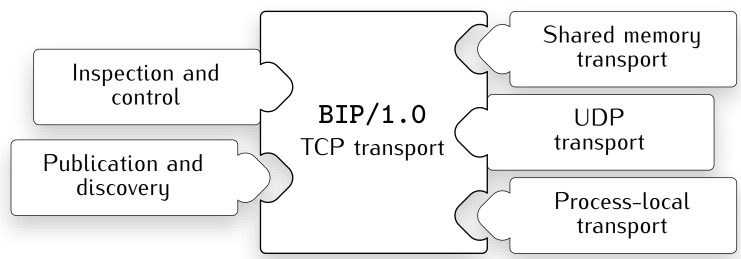


Figure 6.2 . Les modules du protocole BIP/1.0.

Au centre, le cœur de BIP, dont l'objectif est d'être réalisable en une centaine de lignes de code dans tout langage. À gauche, les extensions fonctionnelles spécifiées. À droite, les extensions techniques (mécanismes de transport de données). Les modules « branchés » au centre sont ceux facilement implémentables, et ne requérant pas de dépendances logicielles. Les autres requièrent l'utilisation de DNS-SD (publication et découverte), de APR (transport par mémoire partagée), ou du framework *gmlBIP* (transport local).

Dans *gmlBIP*, les prototypes ci-dessus sont implémentés tels quels, sous la forme méthodes de la classe `ServiceChannel`. Dans le cas le plus simple, la tâche de Laurence se résume donc à écrire quelques lignes de code. Un exemple, en *Tcl*, est donné sur la figure 6.3.

6.1.1.4 Construction de services

gmlBIP fournit une squelette de service, constitué des classes parentes `Service` et `ServiceChannel`. Elles sont chargées de tous les aspects du protocole : encapsulation des messages, ouverture de connections avec une ou plusieurs couches de transport (TCP, UDP, etc), publication et découverte de services, gestion du port de contrôle. La construction d'un service s'apparente donc à l'utilisation d'un framework : Patrick crée une classe dérivée de `Service` et surcharge son constructeur.

La figure 6.4 ci-contre donne un exemple de service fonctionnel minimal `echo`, créé avec *gmlBIP*. Il possède deux canaux, *input* et *output*, et transmet à tous les clients du second les messages reçus sur le premier. Ce service est implémenté en 9 lignes de code.

6.1.1.5 Considérations implémentationnelles

Bien que Laurence puisse réimplémenter aisément `BIP/1.0` quel que soit son environnment logiciel, nous espérons qu'elle utilise une implémentation existante pour communiquer avec les services existants afin de minimiser le seuil fonctionnel de *gmlVision*. Dans cette optique, *gmlBIP* est écrit uniquement en Tcl, et est par conséquent utilisable sur toute plate-forme. Ceci contraint cependant Laurence à utiliser Tcl. Bien que Tcl/Tk soit un environnement très utilisé pour le développement d'applications interactives, il est loin d'être universel.

Pour remédier à ce problème, l'équipe PRIMA a développé, en collaboration avec l'auteur, deux autres implémentations de `BIP/1.0` : O3MiSCID en C++ [Emonet et al., 2006] et jOMiSCID en Java [Reignier et al., 2006]. Ces trois middlewares permettent de communiquer avec les services BIP depuis la majorité des environnements de développements logiciels.

6.1.2 Composants légers

Pour construire des services plus élaborés que l'exemple basique présenté ci-dessus, Patrick doit assembler des composants créés par Stanislas. Il est notable que les canaux

```
# define the event handler
proc _onReceive { peer message_id size data  } {
  puts "received $data from $peer"
}

# create the messaging channel
gmlServiceChannel handle
handle bind <Receive> _onReceive

# connect to the service and send a message
handle connect "gml.echo"
handle send "hello, world"
```

Figure 6.3 . Code minimal permettant de se connecter à un service, utilisant le middleware *gmlBIP*.

de communication (les instances de ServiceChannel) sont également des composants. L'assemblage consiste donc à instancier des composants de communication et des composants de vision, et à router les événements entre composants.

6.1.2.1 API des composants

L'API implémentée correspond à celle proposée au chapitre précédent. Un composant possède deux aspects : d'une part il émet et reçoit des événements, et d'autre part il peut être inspecté et configuré au travers d'une interface commune à tous les composants. Deux classes parentes correspondent à ces deux aspects : EventSource et Configurable.

EventSource représente les sources d'événements. Rappelons qu'elle expose les méthodes bind, unbind et invoke, pour (respectivement) s'abonner à un événement, annuler un abonnement, et déclencher un événement. Les événements qui peuvent être émis par un composant sont définis à l'instantiation. Deux événements sont toujours émis (directement par la classe parente) : <Dispose>, lorsque le composant est détruit, et <Monitor>, décrit dans la section suivante.

L'implémentation actuelle dans *gmlVision* permet l'invocation synchrone ou asynchrone des événements. C'est-à-dire que les *callbacks* associés à un événement émis par un composant peuvent être soit éxécutés immédiatement, soit ajoutés à une file d'attente d'exécution. Actuellement, *gmlVision* est monotâche. En mode asynchrone, la file d'attente est traitée lorsque l'application est en attente. Le mode synchrone est donc préférable pour minimiser la latence. Cette architecture permet l'évolution vers une exécution en parallèle des flux d'exécution des différents composants dans de multiples *threads* d'éxécution afin tirer partie de l'architecture multi-processeurs de la plupart des ordinateurs modernes. Ceci permet de réduire la latence dans le cas où un service possède des composants exécutants des tâches indépendantes.

Configurable implémente l'interface de configuration commune. Elle expose les méthodes configure et cget inspirés de l'API de Tk (la boîte à outils graphique orientée événements distribuée avec Tcl). configure permet d'inspecter les paramètres du composant et de modifier la valeur d'un paramètre ; cget est un accesseur. Les para-

```
# define the class gmlServiceEcho as a service
inherit gmlServiceEcho gmlService

method gmlServiceEcho constructor {} {
  this inherited                      ;# call gmlService constructor
  this configure -class "gml.echo"    ;# define class name

  this newchannel "input"             ;# create channels
  this newchannel "output"

  # attach the _onReceive method to the <Receive> event
  this channel "input" bind <Receive> "$objName _onReceive"
}

method gmlServiceEcho _onReceive { peer message_id size data } {
  # send data back to the service
  this channel "output" send all $data
}
```

Figure 6.4 . Code minimal pour un service fonctionnel, utilisant le middleware *gmlBIP*.

mètres configurables sont définis à la construction par un nom, un type de donnée, et une containte formelle optionnelle (i.e. une expression régulière pour les chaînes de caractères, une formule mathématique pour les valeurs numériques, etc.).

Enfin, les composants implémentent une surcharge de l'API `Configurable` qui émet un événement `<Monitor> configure` lorsque un paramètre est modifié.

6.1.2.2 Monitoring

Afin de permettre à tout composant d'informer le développeur (Stanislas ou Patrick) de son état et de son fonctionnement, un composant peut emettre pendant son exécution des événements `<Monitor>`. *gmlVision* spécifie la forme de ces événements pour les tâches de surveillances rencontrées lors du développement. Cet ensemble est extensible. Les événements possèdent toujours deux attributs : *type* est la classe d'informations de surveillance, et *object* est le composant source de l'événement.

<Monitor> *type*=bitmap *tag [image]*

Le type `bitmap` est utilisé pour suivre des flux vidéo, par exemple les images intermédiaires produites lors de traitements d'image. Le flux d'images est identifié par *tag*, un nom (lisible pour l'humain). Le composant qui émet les évènements `<Monitor>` indique la fin du flux video en envoyant un dernier évènement de type `bitmap` sans spécifier d'*image*.

<Monitor> *type*=draw-rectangle *tag color bounding-box*
<Monitor> *type*=draw-polygon *tag color vertices*
<Monitor> *type*=draw-ellipse *tag color center covariance*

Les types `draw-*` permettent l'enregistrement d'informations discrètes relativement à une image : par exemple les résultats d'un algorithme de détection ou suivi, des régions régions d'intérêt. *tag* indique le nom du flux video sur lequel les rectangles, polygônes ou ellipses doivent être affichés.

<Monitor> *type*=graph-setup *tag min max*
<Monitor> *type*=graph *tag color value*

Pour suivre l'évolution temporelle d'une variable (par exemple : l'énergie moyenne du bruit, indice de confiance), le développeur utilise le type `graph`. `graph-setup` permet de définir les bornes de la valeur correspondante. *tag* désigne un repère et *color* un des graphes dans ce repère.

<Monitor> *type*=configure *option value*

Comme énoncé plus haut, cet événement est automatiquement émis lorsque la valeur d'un paramètre d'un composant est modifiée.

<Monitor> *type*=log *message [level]*

Enfin, `log` permet d'émettre des messages quelconques, par exemple pour fournir des traces informelle sur l'exécution d'un composant. *message* est textuel, et il est associé à *level*, un niveau de criticité.

6.1.3 Inspection de services

Pour rendre observable le fonctionnement d'un service *gmlVision* fournit le composant `ServiceMonitor` qui exploite les API présentés ci-dessus. C'est un composant « client » dans le sens où il s'abonne aux événements (`<Monitor>`) des autres composants mais ne produit pas lui-même d'événements.

Il est asynchrone afin de ne pas perturber le comportement global du composant ou du service qu'il observe. En effet, l'observation d'un service ne doit pas modifier ses performances sous peine de fournir une information erronnée ou déformée. Par exemple, les suivis en vision par ordinateur voient leur robustesse baisser lorsque la latence augmente : hors si le monitoring est invasif, la latence globale du système augmente.

L'affichage graphique et la mise à jour du moniteur doit donc avoir lieu lorsque le processus est en attente d'exécution (i.e. il n'a rien à exécuter). En termes techniques, dans un modèle monotâche comme celui de l'interprêteur Tcl, la mise à jour aura lieu lorsque il n'y a plus d'événement à traiter. Dans un modèle multitâche la *file d'exécution* responsable du moniteur sera associée à une priorité très basse. La conséquence naturelle est que le moniteur aura en général une latence plus élevée (et une fréquence de rafraichissement plus basse) que le système perceptif lui-même.

La classe `ServiceMonitor` est un composant graphique qui s'attache à un service (spécifié à l'instanciation du moniteur). Elle utilise l'API `Configurable` pour déterminer les canaux d'un service et ses paramètres, et l'API `EventSource` pour s'abonner aux événements `<Monitor>` qu'il produit.

L'inspecteur fournit quatre panneaux d'information : les paramètres du service, l'état des canaux de communication, les images et les graphiques produites par le monitoring. Des exemples de ces panneaux sont respectivement présentés sur les figures 6.5 pour le service `grabber.camera`, 6.6 pour le service `tracker.face`, 6.7 et 6.8 pour le service `tracker.correlate` (un sous-service de `tracker.face`).

6.1.4 Conclusion

Sur le plan de l'architecture de *gmlVision*, nous avons implémenté l'ensemble des éléments qui permettent de réaliser l'interface des services de notre boîte à outil en suivant notre conception présentée au chapitre précédent.

Le protocole `BIP/1.0` spécifie l'interopération entre Patrick et Laurence. Notre implémentation de ce protocole, le middleware *gmlBIP*, permet de satisfaire les requis fonctionnels et non fonctionnels énoncés dans le chapitre 4.

Le middleware à composants légers au cœur de *gmlVision* permet à Patrick un développement rapide et modulaire, orienté événements, de services perceptifs. En permet-

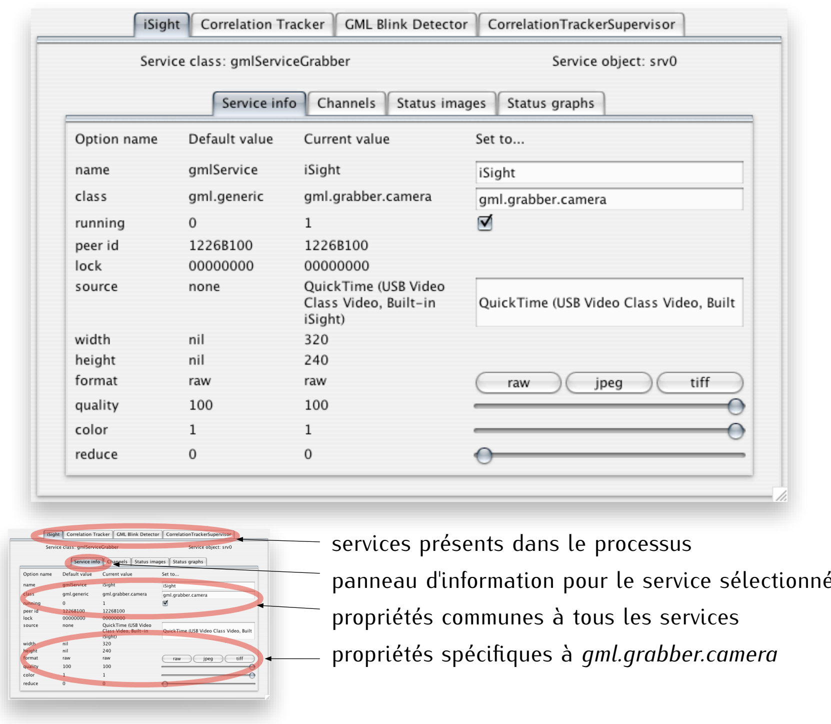


Figure 6.5 . L'inspecteur de *gmlVision* : paramètres du service.

Capture d'écran pour le service `grabber.camera`. L'ensemble des paramètres du services sont visibles. Certains paramètres ne sont pas modifiables (par exemple l'identifiant unique du service), d'autres sont pouvrus d'une interface permettant leur édition.

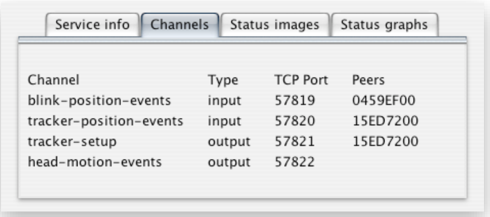


Figure 6.6 . L'inspecteur de *gmlVision* : canaux de communication.

Capture d'écran pour le service `tracker.face`. Sont visible les canaux de communication avec les sous-services `detector.blink` et `tracker.correlate`, ainsi que le canal de sortie. Les trois premiers canaux sont connectés ; la colonne de droite affiche l'ensemble des *peers*.

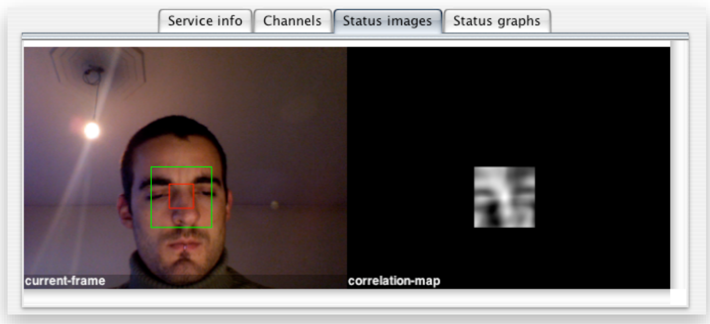


Figure 6.7 . L'inspecteur de *gmlVision* : images intermédiaires et résultats perceptuels.

Capture d'écran pour le service `tracker.correlate`. À gauche, le flux video d'entrée sur lequel est affiché la région détectée (en rouge) et la région de recherche (en vert). À droite, la carte de corrélation pour la zone de recherche ; les points les plus clairs sont ceux où la corrélation est la plus élevée.

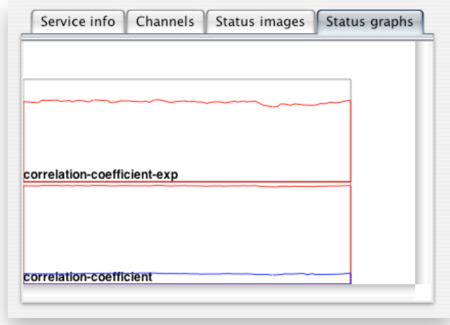


Figure 6.8 . L'inspecteur de *gmlVision* : graphiques.

Capture d'écran pour le service `tracker.correlate`. Les deux graphes affichent l'évolution de l'indice de corrélation entre le modèle et l'objet suivi (ici, le visage de l'utilisateur) en fonction du temps, respectivement avec une échelle logarithmique et une échelle linéaire.

tant aux développeurs de rendre observable le fonctionnement des différents modules, il satisfait un critère d'utilisabilité supplémentaire.

6.2 Services fournis par *gmlVision*

Nous consacrons cette section à la description des services effectivement implémentés dans *gmlVision*.

Le durée de ce doctorat n'était pas suffisante pour implémenter l'ensemble des services décrits aux chapitres précédents. Par exemple, la détection et l'indentification de *tags* [Fiala, 2005, Kato et al., 2000], ou la reconnaissance d'utilisateurs [Zhao et al., 2003] sont des sujets de recherche à part entière, dont l'exploration est ici hors sujet. Cependant, nous pensons qu'il est possible de réutiliser ces résultats de recherche pour construire des services de *gmlVision*. Ces services pourraient être re-développés lorsqu'ils ne satisfont pas, en l'état, les requis que nous avons exprimés au chapitre 3. Ou bien, lorsque l'existant logiciel fournit un service adéquat, le code pourrait être encapsulé sous la forme d'un service *gmlVision*. Par exemple *OpenCV* ou *ARTag* sont de bons candidats pour l'encapsulation de leurs services.

Nous décrivons ici seulement les services qui ont été entièrement implémentés, et ont servi de support à une évaluation : les deux services permettant de construire des interfaces tactiles : `tracker.finger` et `tracker.widgets` ; le suivi de mouvemens de la tête : `tracker.head` ; et les services support `grabber.camera` et `calibrator`.

Nous choisissons donc d'implémenter les services pour lesquels il n'existe pas d'existant satisfaisant, et/ou ceux qui peuvent être implémentés suffisemment rapidement et simplement pour permettre l'évaluation de notre boîte à outils. Nous ne décrirons pas de manière exhaustive l'implémentation de ces services. Nous nous concentrons sur les points qui sont pertinents pour les rendre utilisables.

6.2.1 Le service *Finger Tracker*

Ce service implémente `tracker.finger`. Il s'agit du service le plus abouti de *gmlVision*, car il a été utilisé comme banc d'essai pour concevoir, évaluer et développer la boîte à outils.

Un état de l'art et un premier prototype de `tracker.finger` sont décrit en détail par l'auteur dans un document antérieur à cette thèse [Letessier, 2003]. Un second prototype, le premier à être empaqueté dans un service, est décrit dans [Letessier et Bérard, 2004]. L'implémentation la plus récente, partie de *gmlVision*, est le sujet de cette section. Nous décrirons son évaluation au chapitre suivant.

6.2.1.1 Detection et suivi des doigts

La structure du service actuel `tracker.finger` est synthétisée sur la figure 6.9 page suivante. Les composants utilisés reflètent le déroulement de l'algorithme de suivi. Celui-ci est décomposé en trois étapes classiques, comparables aux phases d'analyse lexicale, syntaxique, et sémantique en traitement du langage.

– segmentation : le premier plan est extrait de l'image (contenant les mains de utilisateurs) sous la forme d'une carte binaire où les pixels sont étiquetés : 1 pour le premier plan, et 0 pour l'arrière-plan.
– détection : un filtre de forme est appliqué à cette carte pour extraire la position des doigts.
– association : les doigts détectés sont mis en correspondance avec les détections antérieures, et les événements de suivi adéquats sont émis.

Rappelons que la principale contrainte sur les choix techniques pour implémenter ces étapes est la latence : le budget de temps imparti à leur exécution n'est que de 20 ms environ. Pour remplir le contrat, le service doit également être aussi autonome que

possible. En particulier, il ne doit pas requérir de réglages pour fonctionner, et encore moins s'il s'agit de réglages nécessitant une expertise en vision par ordinateur.

Segmentation .
Pour l'étape de segmentation, nous avons proposé dans [Letessier, 2003] d'employer une méthode de soustraction de fond. En effet, les autres méthodes (fondées sur un modèle géométriques de la main par exemple) ne sont pas satisfaisantes en termes de latences et contraignent les gestes de l'utilisateur. Ces méthodes ne permettent donc pas de remplir le contrat. Les méthodes de soustraction de fond existantes sont également très diverses. Un état de l'art récent en est présenté dans [Piccardi, 2004]. Nous avons choisi d'utiliser la méthode la plus simple. Un modèle statique du fond est construit en capturant une image à l'initialisation. D'après une métrique de différence d'image (métrique CED basée sur la chrominance dans le cas d'un capteur couleur, différence absolue dans le cas d'un capteur infrarouge), une carte de différence est construite. Cette carte est seuillée uniformément pour produire une image binaire. Le calcul du seuil est automatique d'après un calcul fondé sur l'étude de la forme de l'histogramme. Cette approche de seuillage tombre dans la première catégorie dans la typologie de [Sezgin et Sankur, 2004].

Pour rendre le service autonome, il doit fonctionner lorsque le fond change sensiblement : par exemple lorsque l'éclairage est modifié. Pour ce faire, il serait nécessaire de pouvoir manipuler un modèle du fond plus riche (*eigen-background*, statistiques multimodes par pixel par exemple), ou de pouvoir apprendre ce modèle pendant l'exé-

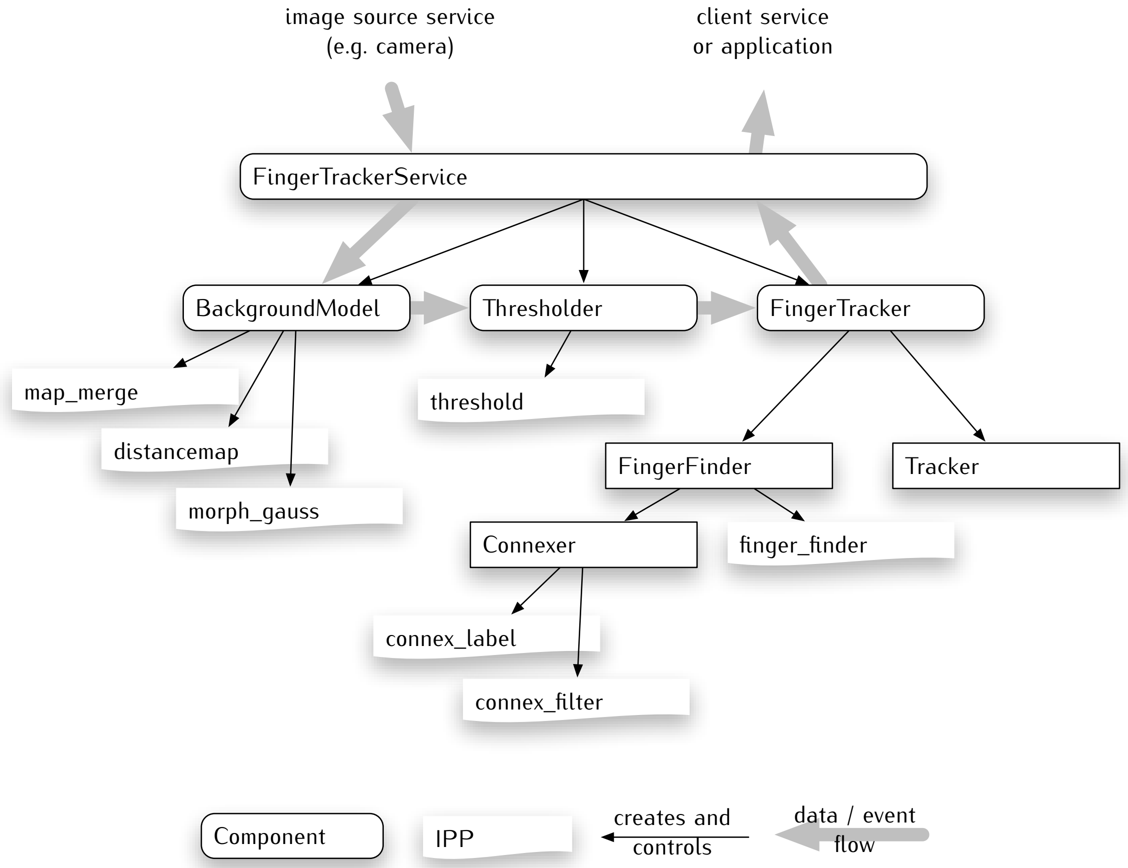


Figure 6.9 . L'architecture logicielle du service de suivi de doigts de *gmlVision*.

Les rectangles arrondis sont des composants ; les autres rectangles sont des classes C++ encapsulées et utilisées en Tcl. Les boîtes sans contours représentent des primitives de traitement d'image, implémentées en Lg (c.f. section B.1 page 187).

cution. La première option étant trop coûteuse en temps de calcul, nous choisissons de re-capturer le fond lorsque le service ne peut plus fonctionner. Ce processus se fait en coopération avec l'utilisateur (grâce aux événements `<Reset>` spécifiés au chapitre précédent).

Cette première étape consomme 75% du temps de calcul propre au service `tracker.finger`, soit typiquement 15 ms (pour une image de définition 320×240 en couleur, 640×480 en infrarouge).

Détection.

La seconde étape est un filtrage de forme chargé de déterminer les positions des extrémités des doigts dans la carte binaire obtenue précédemment. Le formalisme des *filtres de rejet rapides* est introduit dans les documents cités plus haut : il s'agit d'une successions de critères heuristiques appliqués à chaque pixel de la carte binaire. Ils rejettent les pixels candidats le plus rapidement possible, c'est-à-dire que les critères les moins coûteux sont appliqués les premiers. Ce framework est ici particulièrement adapté car des critères fondés sur la géométrie des agents sont faciles à concevoir, à implémenter, et surtout rapides à l'exécution.

Ce filtre de forme fonctionne parfaitement si la segmentation est idéale. Lorsqu'elle est dégradée (trous dans la segmentation, contours rugeux, bruit élevé, arrière-plan segmenté) de fausses alarmes et des détections manquées apparaissent. Elles seront absorbées par l'étape suivante au coût d'un déficit de latence : si par exemple une détection est manquée, la latence double ponctuellement pour le doigt suivi cor-

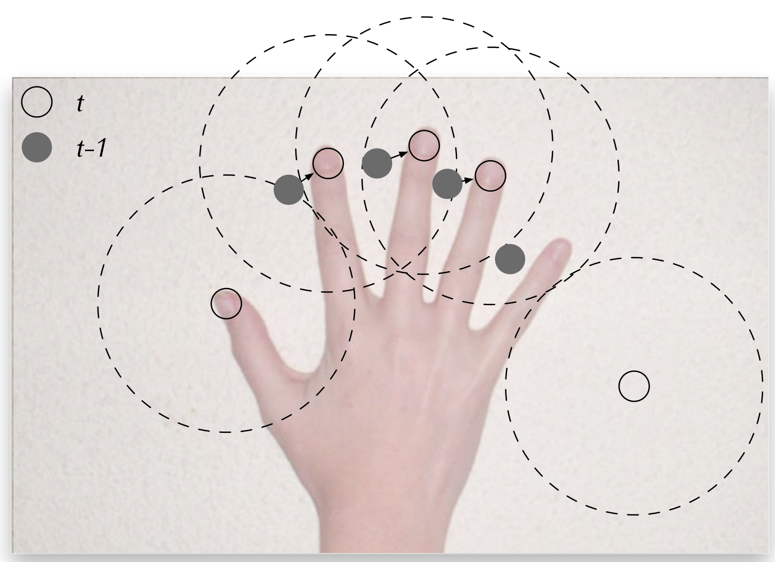


Figure 6.10 . Algorithme d'association de `tracker.finger`.

Cette figure illustre le fonctionnement théorique de l'algorithme de mise en correspondance utilisé pour le suivi dans *gmlVision*. Les cercles pleins représentent les positions des objets suivis à l'instant *t-1*. Les cercles vides représentent les agents détectés à l'instant *t*. Les cercles en pointillés (de rayon *d*) sont les zones de recherche correspondant aux détections.
L'ensemble des points détectés *D* et l'ensemble des points suivis *S* sont mis en correspondance en respectant trois contraintes : tout point de *S* à au plus un correspondant dans *D*, et réciproquement ; la distance entre deux points correspondants est au plus *d* ; la plus grande des distances est minimale. Cette association entre les deux ensemble est unique.
Sur la figure, les associations sont représentés par des flêches (pour l'index, le majeur, et l'annulaire). Lorsque un point de *D* n'a pas de correspondant (le pouce sur la figure, et le point sur la droite), il peut s'agir d'un nouvel objet à suivre ou d'une fausse alarme : une représentation interne est créée, dans l'état *Born*. Lorsque un point de *S* n'a pas de correspondant (le point près de l'auriculaire), il peut s'agir d'une détection manquée ou d'une disparition de l'objet : la représentation interne correspondante passe dans l'état *Zombie*.

respondant. Heureusement, une variance élevée de la latence est tolérable pour les interactions fortement couplées [Watson et al., 1998].

La détection utilise 20% du temps de calcul.

Association .

La dernière étape est celle où les aspect temporels sont pris en compte. Elle consiste à mettre en correspondance les résultats de détection avec l'historique du suivi. Schématiquement, lorsqu'une telle correspondance est trouvée entre un objet suivi et une détection, un événement `<Update>` est émis, étiqueté avec la nouvelle position. Lorsque une détection sans correspondant a lieu, un événement `<Appear>`, étiqueté avec un nouvel identifiant unique, est émis.

À notre connaissance, malgré sa simplicité, l'algorithme n'ayant pas été décrit dans une publication. Nous le représentons donc ici : la figure 6.10 page précédente présente l'association, et la figure 6.11 présente la génération des événements. Remarquons qu'afin d'assurer la stabilité statique du suivi les événements `<Update>` ne sont émis que pour des déplacements supérieurs à 1,5 pixels dans l'image. Ceci correspond au double de l'écart-type observé des mesures pour un doigt fixe, c'est-à-dire que le suivi est stable à 98%. En contrepartie la précision aura pour borne inférieure cette valeur de 1,5 pixels.

L'association utilise les 5% de temps restant.

Finalement, les choix effectués permettent de satisfaire les requis de latence et d'autonomie. L'évaluation de ce service est détaillée au chapitre suivant.

6.2.1.2 Configuration et interface

gmlVision fournit une version « empaquetée » du service `tracker.finger` sous la forme d'une application *FingerTracker.app*. Cette application instancie l'ensemble des services requis. Une capture de son interface est présentée sur la figure 6.12 ci-contre.

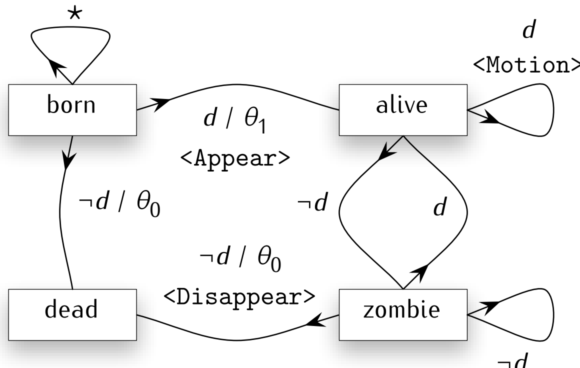


Figure 6.11 . Génération des événements dans `tracker.finger`

Cet automate à états fini décrit le cycle de vie de la représentation interne d'un objet suivi par le service, ainsi que la génération des événements de suivi.
L'algorithme d'association présenté sur la figure précédente crée les objets dans l'état initial *Born*. À chaque instant discret, il leur transmet un évènenemt d si l'objet a été mis en correspondance avec une détection, sinon $\neg d$. Chaque objet maintient un compteur indiquant depuis combien de temps il est dans son état courant.
θ_0 est vrai si le temps écoulé depuis l'entrée dans l'état est au moins t_0, le temps de survie (*time-to-live*). Ceci permet d'absorber les détections manquées : l'objet passe alors dans l'état *Zombie* puis revient en *Alive*. θ_1 est vrai si le temps écoulé depuis l'entrée dans l'état est au moins t_1, le temps de gestation (*time-to-spawn*). Ceci permet d'absorber les fausses alarmes.
Par défaut, t_0 est de 250 ms et t_1 est de 100 ms. Ces valeur ont été choisies empiriquement et conviennent à l'ensemble des prototypes réalisés utilisant `tracker.finger`.

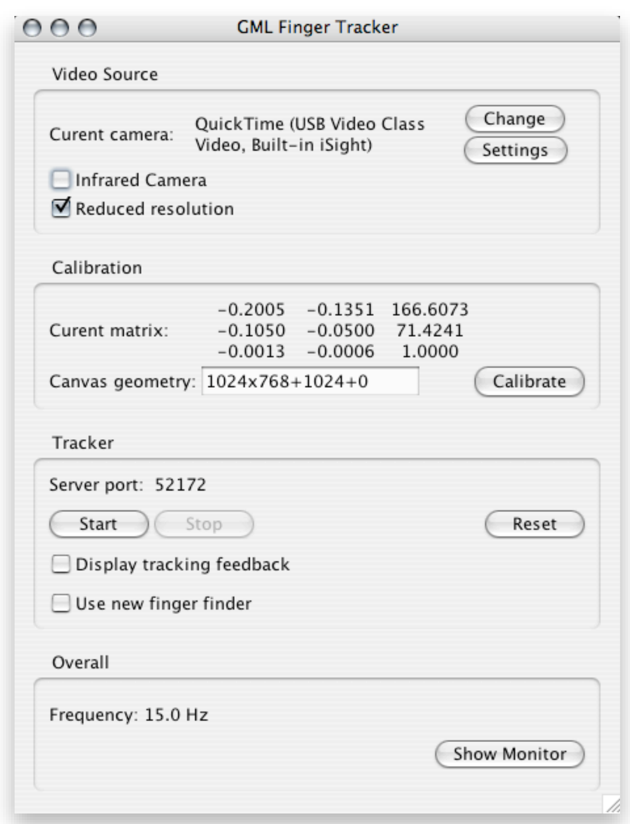


Figure 6.12 . L'interface de l'application *Finger Tracker.app*.

L'application Finger Tracker empaquète trois services : acquisition video, calibrage, et suivi de doigts. Son interface permet le contrôle des services par Caroline. En haut, le composant graphique de `grabber.camera` permet de sélectionner la caméra physique abstraite par le service. Au milieu, `calibrator.manual` permet de spécifier l'étendue de l'interface à calibrer, puis de déclencher le calibrage. Le troisième cadre permet de démarrer ou stopper le service `tracker.finger`. Le bouton `Reset` permet d'effectuer manuellement l'opération de remise à zéro du service (par exemple dans le cas où l'éclairage ne permet plus un fonctionnement acceptable). Enfin, le dernier panneau fournit un indicateur de performance globale et permet d'afficher les inspecteurs de services décrits plus haut dans ce chapitre.

Cette application est adaptée au cas d'usage habituel de Laurence pour les surfaces augmentées : une seule machine exécute le service et l'application interactive et la machine est pourvue de deux dispositifs d'affichage : un écran (de contrôle) et un videoprojecteur. Si l'une de ces conditions n'est pas satisfaite, Laurence ne peut pas utiliser cette application.

Seul le service de capture video doit être configuré : l'utilisateur doit spécifié si la caméra est une caméra couleur ou infrarouge (le service ne peut le déterminer par lui-même). En outre, il est possible d'utiliser un flux video de définition réduite (au détriment de la précision) si le service est exécuté sur une machine dont les performances de calcul sont limitées.

6.2.2 Widgets tactiles

Nous avons décrit l'approche des *widgets sensibles* puis des *SPODs* dans deux publications [Borkowski et al., 2005, Borkowski et Letessier, 2006]. Rappelons que son objectif est de permettre l'interaction digitale sur des surfaces augmentées dans une environnment moins contraint que celui supporté par `tracker.finger`, c'est-à-dire utilisable sur une interface mobile et avec un éclairage non contrôlé. Ce relâchement se fait au détriment de la précision : au lieu de fournir l'équivalent d'une souris à Laurence, on fournit un ensemble de boutons qui peuvent être placés sur l'interface.

L'utilisation du service `tracker.widgets` est décrite sur la figure 6.13. L'idée fondatrice de ce service est de créer des détecteurs d'occlusions élémentaire, performants et robustes aux variations d'éclairage, et de les fédérer en groupes. Un raisonnement

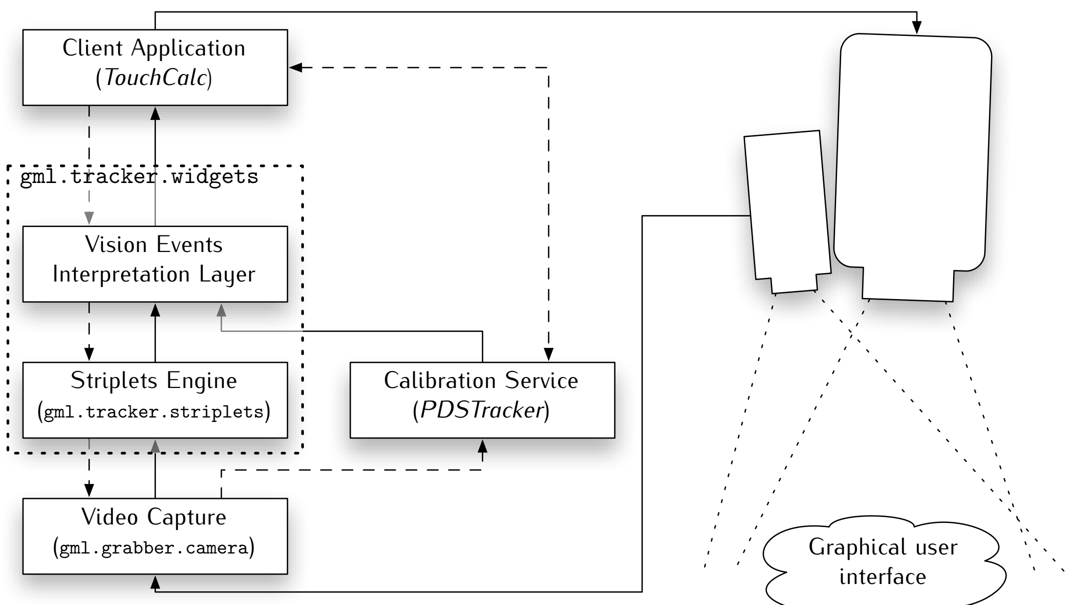


Figure 6.13 . Le service perceptif *SPODS*, qui fournit des widgets digitaux.

Les flèches pointillées représentent les communications à l'initialisation. D'une part, l'application cliente s'abonne auprès du service *VEIL* (couche d'abstraction des événements vision) pour la création de widgets virtuels, en les définissant en coordonnées de l'interface graphique. Ce service demande la création des striplets correspondants. D'autre part, l'application coopère avec le service de calibrage pour établir la correspondance entre les coordonnées de l'interface graphique et le repère image de la caméra. Cette information est transmise au VEIL.
Les flèches en traits plein représentent les communications pendant le fonctionnement. Les images de la caméra sont reçues par le gestionnaire de striplets qui détecte les occlusions et envoie les événements de début et fin d'occlusion au VEIL (`<Hidden>` ou `<Visible>`). Celui-ci fusionne les informations d'occlusion des striplets constituant un widget et génère des événements d'interaction (par exemple `<Button-Down>` lorsque un bouton est déclenché) qui sont transmis au client applicatif.

effectué sur l'état de ses différents membres permet alors de distinguer les occlusions volontaires par un doigt d'autres occlusions parasites.

6.2.2.1 Construction des widgets tactiles

Le service est scindé en deux couches : une couche « bas niveau », le moteur *striplets*, qui analyse l'image entrante pour déterminer des régions occultées par l'utilisateur, et une couche d'abstraction, le VEIL, qui produit des événements de perception.

Couche basse : striplets.

La couche de bas niveau est chargée de notifier la couche supérieure lorsqu'un striplet est occulté ou devient visible en lui transmettant respectivement des événements `<Hide>` et `<Show>`. Les striplets sont des détecteurs d'occlusion élémentaires. En notant d la largeur moyenne d'un doigt (12 mm), il s'agit de rectangles de dimensions d par $3d$. Ils sont associés à une courbe de gain g : $g(x)=-0.5$ si $x<d$ ou $x>2d$, $g(x)=1$ sinon. Étant donnée une image $I(x,y,t)$ a valeurs dans $[0,1]$ (1 pour le blanc, 0 pour le noir), la résponse du striplet à l'instant t est

$$\rho(t) = \frac{1}{3d^2} \sum_{x=0}^{3d} \sum_{y=0}^{d} I(x,y,t) g(x) \quad \in [-1, 1]$$

Placés sur un fond uni un *striplet* produit donc une réponse positive lorsque son centre est seul occulté par un objet de la taille d'un doigt et une réponse faible ou négative sinon. En particulier, il produit une réponse quasi nulle s'il est entièrement occulté par un objet uniforme, ou entièrement non occulté. Il est possible de décider si un *striplet* est occulté ou non en fixant un seuil θ : lorsque $\rho>\theta$ on émet un événement d'occlusion. Le fonctionnement d'un striplet est résumé par la figure 6.14.

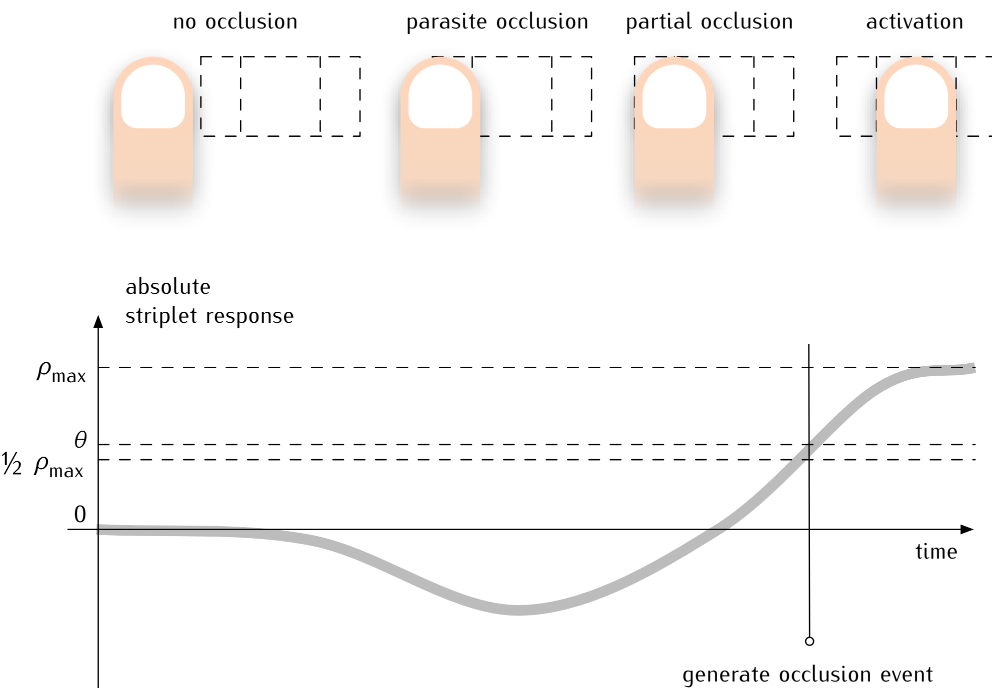


Figure 6.14 . Détection des occlusions pertinentes avec les *striplets*.

Cette figure représente un scénario possible d'occlusion d'un *striplet*. Un doigt se déplace de gauche à droite, suivant l'axe du striplet. Les images en haut représentent les positions relatives du doigt et du striplet à des instants critiques. Le graphique représente l'évolution théorique des variations de la réponse du striplet en fonction du temps. La barre verticale indique l'instant où un événement d'occlusion est émis. ρ, la résponse du *striplet*, et θ, le seuil de déclenchement, sont décrits dans le texte principal.

Par souci d'autonomie, dans *gmlVision*, le seuil θ est déterminé automatiquement pour chaque striplet au moment de sa création. L'étude de la réponse pendant les premières images reçues (1 seconde de vidéo, soit 25 ou 30 images en général) permet de déterminer la médiane μ et l'écart-médian σ de la réponse, c'est-à-dire des évaluations robustes de la moyenne et du bruit de la mesure. Idéalement, si la surface où le striplet est placé est uniforme, et si la mesure est non bruitée, μ et σ sont nuls. Ces valeurs sont ensuite supposées être constantes. Elles permettent de fixer un seuil selon la formule :

$$\theta(t) = \frac{1}{2}\left(\max_{\tau=t-T}^{t} \rho(\tau) + \mu\right)$$

Informellement, ceci correspond à l'activation du striplet lorsque sa réponse compensée est supérieure à la moitié de la plus grande réponse récemment observée. La fenêtre glissante de largeur T (fixée à 1 seconde) permet de compenser les variations des illuminations relatives de la surface et de l'agent dues à des changements d'éclairage. Pour éviter les fausses alarmes, on effectue un seuillage à hysteresis : un événement `<Hide>` est généré lorsque $\rho > \theta + 2\sigma$, et un événement `<Show>` lorsque $\rho < \theta - 2\sigma$. Nous avons observé des valeurs typiques pout μ est de l'ordre de $\pm 5\%$, σ de 1%, et θ de 30%.

Remarquons que les *striplets* ne sont pas visibles pour Laurence : seules les fonctionnalités des *SPODs* le sont.

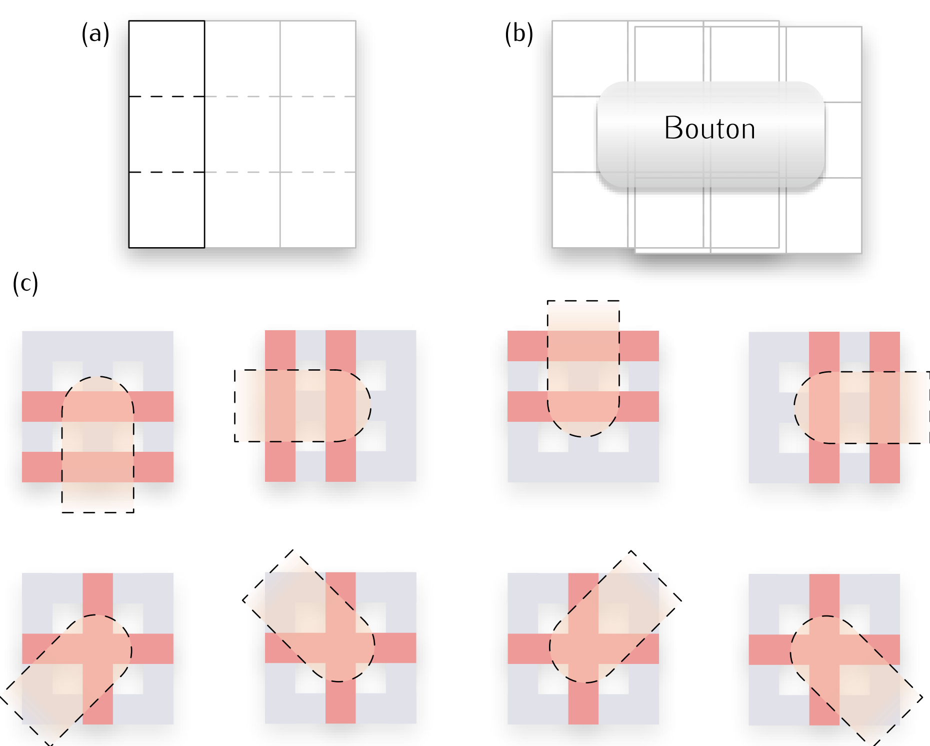


Figure 6.15 . Boutons sensibles dans `tracker.widgets`.

Les boutons sont un exemple de SPOD. L'image *(a)* montre la disposition des six *striplets* formant un bouton : trois horizontalement, trois verticalement. Sur *(b)*, on voit comment il est possible d'assembler deux boutons SPOD pour recouvrir un widget bouton de l'interface graphique. En *(c)*, les configurations possibles d'un bouton tactile activé : soit les deux striplets centraux sont occultés, soit un seul striplet central et un seul striplet parallèle. Ces configurations correspondent à huit angles d'incidence d'un doigt. Les striplets sont représentés plus « fins » qu'en réalité par souci de lisibilité.

Couche haute : SPODs.

Les *striplets* ne peuvent pas être utilisés directement pour créer une interface tactile, ceci pour plusieurs raisons : leur taille est fixe (36 par 12 mm), il sont anisotropes (ils ne peuvent être activés que par un doigt grossièrement perpendiculaire à leur axe), et ils sont sans mémoire (leur activation instantanée peut être involontaire). Les SPODs (*Simple Pattern Occlusion Detectors*) sont des objets logiciels. Chaque SPOD est constitué d'un ensemble de striplets et émet des événements perceptifs en analysant leurs motifs d'activation. En d'autres termes, un SPOD utlise des règles heuristiques sur l'activation de ses composantes pour déterminer s'il est lui-même activé. Selon l'assemblage de striplets utilisé, un SPOD peut servir à implémenter plusieurs types de widgets. Lors de la conception de `tracker.widgets` nous en avons implémenté trois : le bouton poussoir, le *slider* (barre de défilement), et le trackpad (zone tactile). Nous décrivons ici uniquement le bouton car il permet de réimplémenter les autres.

Sur la figure 6.15 ci-contre est présenté l'assemblage de striplets permettant de construire un bouton. Afin de rendre la détection quasi-isotrope, deux règles d'activation sont définies. Grâce à elles, huit orientations possibles du doigt peuvent donc activer le bouton : par l'un des côtés ou l'un des coins. Afin d'éviter les fausses alarmes, un bouton SPOD activé ne transmet un événement `<Appear>` que s'il le reste pendant une durée déterminée (de la même manière que le temps de gestation t_1 dans `tracker.finger`), réciproquement pour `<Disappear>`. Ce délai est fixé à 100 ms.

Enfin, pour compenser la taille fixe des striplets, donc des boutons SPOD, il est possible de superposer plusieurs SPODs pour recouvrir la surface d'un widget de l'interface graphique. Plusieurs SPODs peuvent éventuellement partager des striplets, afin de réduire le coût de calcul. Sur la figure, le widget est recouvert par deux SPODs : en plus de servir de bouton poussoir, il est donc capable de préciser si l'utilisateur a posé son doigt à droite ou à gauche du bouton. En généralisant, on constate qu'il est possible d'utiliser une chaîne de bontons pour construire un *slider*, capable de fournir une information de position à une dimension ; ou une matrice de boutons pour construire un *trackpad*, qui fournit les coordonnées du doigt sur la surface.

Laurence ne doit pas avoir à gérer la superposition de plusieurs SPODs pour construire un widget sensible ; `tracker.widgets` devront donc masquer la structure d'un widget et gère la fusion des événements `<Appear>` et `<Disappear>` de façon à ce que l'utilisateur ne perçoiven qu'un widget logique, là où le système utilise éventuellement de nombreux SPODs.

En augmentant le nombre de striplets et en rapprochant les SPODs utilisés pour construire un trackpad, il est possible d'augmenter la précision du widget obtenu. À la limite on pourrait donc envisager de remplacer `tracker.finger` par `tracker.widgets`. Cependant le coût (en temps de calcul) augmente linéairement avec le nombre de striplets traité, alors que `tracker.finger` a un coût constant pour une taille d'image donnée.

Un exemple d'application construite en utilisant `tracker.widgets` en entrée est présenté sur la figure 7.9 page 155.

6.2.2.2 Configuration et interface

Comme précédemment, nous avons empaqueté `tracker.widgets` dans une application *VirtualWidgets.app*. Son interface est très semblable à celle du FingerTracker. Cependant, au contraire de `tracker.finger`, `tracker.widgets` est autonome. Il n'y a donc pas de bouton *Reset* dans l'interface graphique du service (il n'est pas nécessaire d'initialiser manuellement le service).

Le service peut fonctionner avec une caméra couleur ou avec une caméra infrarouge : par souci de simplicité, le panneau de calibrage manuel est toujours affiché, même lorsque le calibrage automatique est possible.

6.2.3 Suivi de visage

Ce troisième service implémente `tracker.face`. Bien qu'il n'ait pas subi autant de tests que les deux services décrits précedemment, il est particulièrement pertinent de le décrire car il met en œuvre les mécanismes d'interconnection de services.

Il se décompose en trois services en interaction :

- `tracker.aspect`, un suivi récursif précis et faiblement bruité, mais non autonome, à latence faible ;
- `detector.blink`, un méthode de détection permettant d'initialiser le suivi, autonome, et a latence élevée ;
- `tracker.face`, un superviseur capable d'initialiser le premier d'après les résultats du second, et de transmettre les informations de suivi au client.

Seul le service superviseur est exposé pour l'utilisateur (Laurence). Cette structure est résumée sur la figure 6.16.

Suivi récursif.

Nous appelons « suivi récursif » un algorithme de vision qui recherche la nouvelle position d'un agent en fonction de la position précédente, typiquement dans son voisinage. Le problème de l'initialisation (*bootstrap*) est de déterminer la position initiale de l'agent. Ce type d'algorithme est répandu en vision. C'est l'approche adoptée par le tracker de points d'intérêt KLT [Tomasi et Kanade, 1991] ou par le blob-tracker Camshift [Bradski, 1998].

`tracker.aspect` est un suivi par corrélation croisée normalisée. Étant donnée une imagette cible T une image d'entrée I, cet algorithme cherche T dans I, c'est-à-dire la position de I à laquelle T ressemble le plus, suivant une métrique particulière. Pour plus de détails le lecteur peut consulter [Crowley et Berard, 1997]. Cet algorithme est adapté au besoin qu'implémente `tracker.face` : il est très rapide, robuste aux changements d'illumination globaux. Il est peu robuste aux rotations mais celles-ci sont peu susceptibles d'avoir lieu dans les situations d'interaction envisagées (décrites page 53).

Détection initiale.

Pour initialiser le suivi, `detector.blink` fournit à `tracker.aspect` la position et les dimensions d'une imagette dans l'image courante en détectant les clignements d'yeux, volontaires ou non, de l'utilisateur. Ce service s'appuie sur les résultats

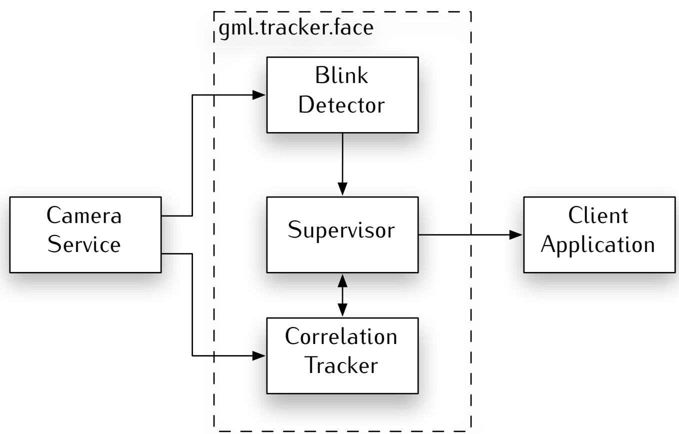


Figure 6.16 . Architecture du service `tracker.face`.

Le service `tracker.face` est exposé par le composant superviseur. Celui-ci utilise les informations issues d'un service de détection de clignement d'yeux (`detector.blink`) pour initialiser un service de suivi par corrélation (`tracker.aspect`). Ainsi, le service obtenu est autonome.

de [Bérard et Coutaz, 1996], [Grauman et al., 2001], [Kawato et Tetsutani, 2002], et [Ohno et al., 2003]. Le clignement des yeux naturel a une durée de 100 ms, ce qui constitue une borne inférieure de la latence de ce service.

Pour chaque image, le service calcule une carte de distance avec l'image ayant été capturée 100 ms plus tôt, la seuille automatiquement, et effectue une analyse en composante connexe (en utilisant les mêmes composants que `tracker.finger`). Un ensemble de filtres anthropomorphiques est ensuite appliqués pour réduire le nombre de composantes : celles de taille ou de proportions abbérante, et celles qui présentent peu de symétrie radiale. Une autre série de filtres est appliquée à toute les paires possibles de composantes restantes : sont éléminiées celles dont les proportions ne correpondent pas, celles qui sont trop éloignées d'une horizontale commune, et celles qui sont trop éloignées ou trop proches vis-à-vis de leurs dimensions. Ces filtres sont indépendants de l'échelle de la vue. Finalement, s'il reste exactement une paire, le service suppose qu'il s'agit d'yeux qui ont cligné, et émet leur position.

Le service `tracker.face` est connecté aux deux autres. Tant que `tracker.aspect` fonctionne, il ignore les résultats de `detector.blink`. Par contre, si `tracker.aspect` ne fonctionne plus (parce qu'il a perdu le visage, ou parce qu'il vient de s'initialiser), `tracker.face` calcule les dimensions d'une imagette à partir des coordonnées du clignement détecté et les fournit à `tracker.aspect`.

Ici, les deux sous-services pourraient être remplacés par d'autres méthodes. Par exemple nous avons effectué des tests prélimianires en remplaçant `detector.blink` par la routine de détection de visage de *OpenCV*. Sa latence est plus élevée (200 ms de temps de calcul pour une image de 320×240 pixels, et statiquement instable), mais il fournit une information de position sans devoir attendre le prochain clignement d'yeux. De même, le suivi par corrélation pourrait être remplaçé par `tracker.color`, qui est insensible à l'orientation du visage.

6.2.4 Services de calibrage

Dans le cadre des surfaces interatives, le calibrage géométrique est l'opération qui permet d'obtenir une fonction de transfert entre les coordonnées d'un point M' de l'interface graphique et un point M de la vue caméra. En supposant que la caméra est « idéale », c'est-à-dire obéit au modèle sténopé, cette fonction est modélisable par une transformation projective (ou homographie) P de dimension 3, telle qu'en coordonnées homogènes, $M' = PM$. La figure 6.17 page suivante résume une méthode pour déterminer H à partir de correpondances connues entre points de l'interface et points de la caméra.

Comme nous disposons d'une méhode connue pour établir l'homographie, la difficulté réside dans la déterminations de correspondances entre points connus dans l'interface et points perçus dans l'image caméra.

6.2.4.1 Calibrage manuel

Pour `calibrator.manual`, l'établissement des correspondances est trivial. L'interface graphique du service, destiné à Caroline, affiche la video issue de la caméra. Le service (par le biais de sa connexion à une instance du service `display`) affiche successivement des disques blancs sur fond noir sur l'interface, et il est demandé à Caroline de cliquer sur les centres de ces disques sur la video. Une fois suffisemment de correspondances établies le service termine le calcul et émet un événement `<Matrix>`.

6.2.4.2 Calibrage automatique

Dans le cas des deux autres services, un processus automatique est mis en place. Il consiste à afficher un mire sur l'interface et à détecter la position des $N \times N$ points de cette mire. Nous notons $\mathbf{M'}$ l'ensemble des points de la mire, et $\mathbf{M}$ l'ensemble des points détectés. Pour déterminer $\mathbf{M}$, les opérations suivantes sont effectuées :

– capture d'une image avant et après l'affichage de la mire ;
– calcul d'une carte de distance entre les deux captures (différence des valeurs absolues) ;
– seuillage automatique (en utilisant le même procédé que page 128) ;
– débruitage de la carte seuillée par érosion morphologique ;
– analyse en composantes connexes, et calcul des centres des N plus grandes composantes, avec une précision inférieure au pixel.

Après cette phase de détection, il reste à mettre en correspondance les points détectés et les points connus, c'est-à-dire établir une bijection entre M et M'. Pour `calibrator.assisted`, la mire est une grille de points affichée par un client et le service est informé de la position de ces points. En outre, le contrat du service spécifie que la vue caméra et l'interface sont presque alignés, c'est-à-dire que la distorsion est faible. En d'autres termes, les relations entre les abscisses et les ordonnées dans M' sont préservées dans M. D'après cette supposition, le service trie M suivant les ordonnées des points. Les N premiers points de M correspondent donc à ceux de la première ligne de M', les N suivant à ceux de la seconde ligne, etc. Les groupe de N points de M sont ensuite triés suivant leurs abscisses. La bijection est alors établie, le service peut terminer le calcul.

Pour `calibrator.automatic`, la distortion peut être arbitraire. Typiquement, le *keystoning* de la vue caméra peut être élevé. Ou bien la caméra peut avoir un axe optique orthogonal à l'interface mais ne pas être alignée. Quelle que soit la mire, il n'est pas possible de faire de supposition sur l'ordre des points. Le service utilise donc une mire

$$\begin{bmatrix} x_1 & y_1 & 1 & . & . & . & -x_1x'_1 & -y_1x'_1 & -x'_1 \\ . & . & . & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 & -y'_1 \\ x_2 & y_2 & 1 & . & . & . & -x_2x'_2 & -y_2x'_2 & -x'_2 \\ . & . & . & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 & -y'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & . & . & . & -x_nx'_n & -y_nx'_n & -x'_n \\ . & . & . & x_n & y_n & 1 & -x_ny'_n & -y_ny'_n & -y'_n \end{bmatrix} \times \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_9 \end{bmatrix} = \begin{bmatrix} dx_1 \\ dy_1 \\ dx_2 \\ dy_2 \\ \vdots \\ dx_n \\ dy_n \end{bmatrix}$$

$$\mathbf{A} \times \mathbf{P^*} = \mathbf{D}$$

Figure 6.17 . Formule de calibrage géométrique.

Pour plus de détails et une démonstration formelle le lecteur est invité à consulter [Criminisi et al., 1997]. La relation ci-dessus établit le lien entre la matrice de calibrage A, la transformation projective P, et le vecteur-erreur D. A est définie en fonction des coordonnées x et y des points M, et x' et y' des points M'. D est le vecteur des différences entre coodonnées des points de l'interface et des points reconstruits ; c'est-à-dire que dx est l'abscisse de M et dy son ordonnée. P^* est une version « aplatie » de P, c'est-à-dire dont les lignes ont été concaténés pour former un vecteur.
La matrice A est obtenue en simplifiant le système d'équations $dM = M' - PM$, pour tous les couples M, M'.
L'objectif de minimiser la norme euclidienne de D, c'est-à-dire l'erreur quadratique commise en approximant la transformation de M en M' par la transformation projective P. La valeur de P optimale est le vecteur propre correspondant à la plus petite valeur propre de $T(A)A$, où T est l'opérateur de transposition.
Cette opération est possible dès que 4 correspondances sont établies, puisque P possède 8 degrés de liberté.

anisotrope (représentée sur la figure 6.18). Après la detection des centres des points de la mire, il détermine les cinq points de l'enveloppe convexe de cet ensemble. Comme il existe cinq correspondances possibles entre l'enveloppe de la mire et celle des points détectés, le service test les cinq hypothèses correspondantes. Pour chacune, une homographie grossière h est calculée à partir des cinq correspondances. L'application de h à M produit un ensemble M*, qui est mis en correspondance avec M' suivant là même technique que pour `calibrator.assisted`. L'erreur de calibrage est calculée pour chaque hypothèse. Comme une seule est valide, l'hypothèse et la bijection produisant l'erreur la plus faible sont retenues. Enfin, le service calcule l'homographie finale P, cette fois en utilisant tous les points de M et M'.

Afin d'éviter les échecs de calibrage, tous les services calculent l'erreur de reconstruction, définie comme la plus grande distance entre *M'* et *PM*. Si cette erreur est plus grande que la moitié du pas de la grille de calibrage, le service considère que le calibrage a échoué, et en notifie le client.

Autres algorithmes.

L'algorithme de calibrage automatique est particulièrement complexe. Dans la littérature, on retrouve le calibrage automatique sous une forme simplifiée : soit les points de la mire sont affichés successivement, ce qui évite le problème de mise en correspondance ; soit un simple rectangle est utilisé comme mire, et ses quatres coins comme points de référence [Sukthankar et al., 2000]. Le défaut de la première solution est sa latence. En effet, il est nécessaire de synchroniser les diférentes captures avec les cycles d'affichage, qui ont une latence variable. D'après notre expérience, pour une mire « standard » de 15 points, notre approche permet le calibrage en 100 ms, contre environ 2500 ms pour l'affichage itératif. La seconde solution manque de précision ; en utilisant cette méthode, nous avons observé des erreurs de calibrage typiques de 5 mm, contre environ 0.5 mm avec notre méthode (pour 15 points).

6.2.4.3 Interface utilisateur

Le service `calibrator.manual` dispose d'une interface qui peut être encapsulée dans celle d'un paquetage de services. C'est le cas pour l'interface de `tracker.finger`

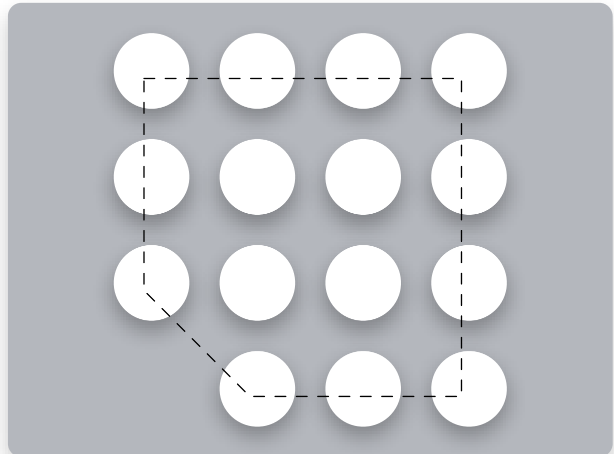


Figure 6.18 . Mire de calibrage utilisée par `calibrator.automatic`, de dimension 4×4.

La mire est projetée de manière à ce que ses points constrastent au mieux avec le fond, typiquement en blanc sur fond noir. Le point d'indice (0,0) n'est pas affiché, afin d'orienter la mire ; la mire possède donc *N*×*N-1* points.
En pointillés, l'enveloppe convexe de la mire. Lors de la détection, cette enveloppe peut contenir des points intermédiaires superflus, qui sont supprimés en utilisant un critère de colinéarité.

et tracker.widgets, décrits plus haut. Les deux autres services sont autonomes du point de vue de Caroline, ils ne possèdent donc aucune interface.

6.2.5 Services video

Nous avons implémenté trois services qui produisent des flux video dans *gmlVision*.

Les deux premiers, grabber.camera et grabber.offline constituent la fondation de *gmlVision*. En effet, l'abstraction du matériel d'acquisition video est un point critique en vision par ordinateur. D'une part il s'agit de la principale source de **latence** dans un système perceptif. D'autre part les API permettant d'accéder aux images capturées et de contrôler la capture et la caméra sont habituellement complexes, souvent obscures, et spécifiques à chaque plate-forme.

Pour implémenter grabber.camera, *gmlVision* fournit une interface au niveau C et au niveau Tcl qui permet d'accéder à ces fonctionnalités de manière uniforme quelles que soient la plate-forme, l'API de capture, et la caméra. Pour référence, la couche d'abstraction en C des différentes API représente à elle seule 6000 lignes de code.

Ce service de capture est capable d'émettre des images avec le protocole BIP en utilisant les transports TCP, SHM, et Local. Un prototype de communication sur UDP (pour des applications de type videoconférence) est à l'étude, mais il n'est pas développé car il ne correspond pas à un besoin de perception articielle, coeur de notre étude. grabber.camera permet en outre, via son interface de contrôle ou son moniteur, de piloter les paramètres de la caméra et de l'acquisition. Sont contrôlables au minimum les fonctions de gain automatique et de balance des blancs, et, selon disponibilité, l'iris, le temps d'obturation, le zoom, etc. Il permet aussi de décimer l'image d'entrée si les contraintes de performance l'exigent.

Le service grabber.offline lit une vidéo stockée sur disque et fournit ses images à des fins d'expérimentations. Il offre une version réduite de l'API de grabber.camera : il est en effet impossible d'effectuer *a posteriori* des réglages sur les propriétés de l'acquisition. Par contre, il possède des paramètres supplémentaires spécifiant le fichier vidéo à jouer, la fréquence à laquelle le jouer (si différente de la fréquence native), et s'il doit recommencer au début du fichier lorsqu'il est terminé.

Enfin, filter.lighting est un service d'analyse d'un flux video brut. Son objectif est la production d'une carte de chaque image du flux où les zones impropres au traitement sont indiquées. Le produit de ce service peut en particulier être fournit aux autres services perceptifs afin qu'ils ne traitent pas ces zones pour des raisons de performance et de robustesse. Un pixel de l'image courante et étiqueté comme invalide s'il satisfait au moins l'une des trois conditions suivantes :

- il est sous-exposé, i.e. sa valeur moyenne (calculée à l'aide d'une moyenne courante) est en-deça d'un seuil a ;
- il est sur-exposé, i.e. sa valeur moyenne est supérieure à $1\text{-}a$;
- son bruit moyen, estimé par la moyenne de ses variations entre frames successives, est supérieur ou égale à b.

Nous utilisont la valeur de 5%, déterminée empiriquement, pour a et b.

6.2.6 Conclusions sur les services implémentés

Bien que nous ayons spécifié un ensemble de services recouvrant fonctionnellement les besoins de vision pour l'interaction identifiés au chapitre 2, nous n'avons pas implémenté l'ensemble de ces services. Néanmoins, les services implémentés correspondent à leurs spécifications, telle que données en 5.3. En particulier, ils satisfont les deux contraintes de qualité de service les plus importantes pour les utilisateurs : la latence et l'autonomie.

Si le temps limité a été un facteur, les services que nous avons choisi de ne pas implémenter ne l'ont cependant pas été par hasard. Par exemple, la fonctionnalité de

`tracker.surface` a été implémentée dans l'équipe PRIMA dans le cadre du projet ContAct. La fonctionnalité du service `tracker.color` a été implémenté pour la *Table Magique* dans l'équipe IIHM. Enfin, *ARTag* [Fiala, 2005], projet dont le code est disponible, fournit la fonctionnalité de `tracker.tags`.

L'effort restant consiste essentiellement à empaqueter ces implémentations dans des composants *gmlVision*, puis dans des services.

6.3 Conclusions

Du point de vue de Patrick *gmlVision* est un framework. L'implémentation fournit un squelette de service qu'il doit « habiller » avec des composants de communication (canaux) et des composants de perception (construits par Stanislas). Il connecte ensuite ces composants les uns aux autres *via* une API orientée événements, permettant au flux de l'information de se propager jusqu'à l'utlisateur. Enfin, chaque service est empaqueté dans une application qui fournit une interface graphique permettant, le cas échéant, de le configurer.

Du point de vue de Laurence, *gmlVision* est une bibliothèque constitué de l'ensemble de ces applications exécutables qui fournissent un ou plusieurs services. Sa tâche est donc de choisir, parmi ces applications, celle qui répond à son besoin. Ensuite, connaissant la spécification du service correspondant, elle écrit le code permettant de s'interfaçer avec ce service — éventuellement en utilisant *gmlBIP* — et de traiter les événements perceptifs pour réaliser des techniques d'interaction.

La réalisation de *gmlVision* est incomplète : certains services sont manquants, ou non finalisés (nous ne les avons pas décrits ici). Cependant, elle est suffisante pour procéder à des expérimentations visant la validation de notre approche : nous disposons d'une structure et de mécanismes d'interopération pour intégrer des services à une système perceptif éventuellement hétérogène. Surtout, nous disposons de services perceptifs qui permettent la création ou la reproduction de systèmes interactifs que nous pouvons ensuite évaluer.

7 Déploiement, validation et évaluation

> *« Technology designers must select solutions that meet specific user requirements. Although some computer vision techniques may be relevant, they are dificult to assess, since most are evaluated only in terms of their absolute accuracy. (...) Assessments of computer vision techniques should include cases with feedback to the user. »*
>
> *[Eisenstein et Mackay, 2006]*

L'un des standards de l'évaluation logicielle, CMMI [Chrissis et al., 2003] donne une définition de la validation logicielle : « La validation confirme que le produit fournit satisfait l'usage auquel il est destiné. En d'autres termes, la validation assure que le *bon* produit a été contruit. » Un autre standard, ISO 25000, founit un cadre de référence qui définit certains critères d'évaluation logicielle. D'après [Borkowski, 2006], il définit la qualité à l'utilisation (*quality in use*) par la « mesure dans la quelle un produit satisfait les besoins d'utilisateurs spécifiques pour atteindre un but spécifié avec éfficacité, productivité, sûreté et satisfaction, dans des contexte d'usages eux-mêmes spécifiés. » CMMI comme ISO 25000 sont des documents volumineux qui définissent avec précision les termes du domaine (efficacité, productivité, etc.) ainsi que des objectifs et des méthodes de production logicielle dans le cadre d'un développement par une entreprise. L'application à la lettre de l'un ou l'autre standard constituerait un travail hors du sujet de cette thèse, et d'ailleur trop conséquent pour y être traité. En particulier, ils sont destinés à être employés pour évaluer un produit industriel plutôt qu'un prototype de recherche tel que *gmlVision*.

Il n'existe cependant pas de méthodologie standard, ni même résultats de recherche concernant l'évaluation de l'API d'une bibliothèque logicielle en tant qu'interface utilisateur. Plusieurs auteurs font ce constat [Klemmer et al., 2004, Appert et Beaudouin-Lafon, 2006] : « l'évaluation d'une boîte à outils est un problème bien connu mais mal résolu. » Les métriques quantitatives connues (nombre des lignes de code de la boîte à outils, taille en mémoire) ne sont pas pertinentes. Le consensus est alors de faire un test d'utilisabilité de la boîte à outils, en évaluant quantitativement les sujets sur des tâches classiques.

Malheureusement, nous n'avons pas eu la possibilité de faire un tel test d'utilisabilité pour *gmlVision*. D'une part, le temps nous à manqué. D'autre part il n'existe pas encore de « tâches classiques » sur lesquelles évaluer des sujets dans notre domaine émergent, comme nous l'avons évoqué en 4.1.2.3 page 66. Notre évaluation sera ici nécessairement informelle.

Nous devons valider et évaluer *gmlVision*, et à travers elle, notre approche. Notre objectif est d'avoir une démarche centrée utilisateur tout au long de la réalisation de la boîte à outils. Nous proposons donc les définitions suivantes de la validation et de l'évaluation :

La **validation** consiste à montrer l'**utilité** de la boîte à outils pour chacune des classes d'utilisateurs, c'est-à-dire la satisfaction de l'aspect fonctionnel de leurs besoins. La validation est une mesure binaire.

L'**évaluation** est la démonstration de l'**utilisabilité** de la boîte à outils. L'évaluation consiste à mesurer l'adéquation du produit avec les besoins non fonctionnels. L'évaluation est une mesure de qualité complexe possédant de nombreux axes. Selon les axes la mesure est continue ou discète, quantitative ou qualitative, objective ou subjective.

Le travail doctoral présenté ici a été exploratoire : les critères d'utilité et d'utilisabilité pertinents pour notre boîte à outils ont, pour certains, émergé lors des cycles successifs de conception et de développement. C'est pourquoi nous présentons, dans la première section de ce chapitre (7.1), un historique du déploiement de systèmes interactifs utilisant *gmlVision*. À elle seule, la possibilité de construire de tels prototypes constitue une validation (telle que définie ci-dessus) de l'approche proposée dans les chapitres précédents.

L'évaluation de *gmlVision* est l'objet des deux sections suivantes (7.2 page 159 et 7.3 page 168). La première est consacrée à l'évaluation individuelle des services fournis par la boîte à outils qui ont effectivement été utilisés pour réaliser des systèmes interactifs. Cette section correspond à l'application des requis fonctionnels présentés au chapitre 3. La seconde traite de l'architecture orientée services que nous avons présentée indépendemment du cadre de la bibliothèque. Cette section correspond à l'application des requis structurels présentés au chapitre 4.

7.1 Déploiement de prototypes

La réalisation de *gmlVision* a dès ses débuts été guidée par les besoins utilisateurs. La première motivation a été de permettre la manipulation au doigt sur la *Table Magique* utilisée dans le projet de recherche FAME (*Facilitating Agent for Multi-Cultural Exchange*, IST-2000-28323). Peu de temps après, le besoin d'une interaction naturelle est apparu dans le cadre du projet ContAct (*Context management for pro-Active computing*, RNTL/Proact).

gmlVision est bâtie sur les fondations de *Tclvision*, une boîte à outils de vision pour l'interaction de type classique (au sens de la typologie du chapitre ??). *TclVision* a notamment été utilisée pour développer la *Table Magique* [Bérard, 2003]. La structure générale de la boîte à outils (implémentation en C/C++ et abstraction en Tcl des primitives de traitement d'image et de vision) a été conservée, mais toutes les fonctionnalités décrites dans ce document (traitement d'image, vision, composants, services) ont été conçue et écrites pour *gmlVision*.

La figure 7.1 résume l'évolution de notre boîte à outils en retraçant l'historique des

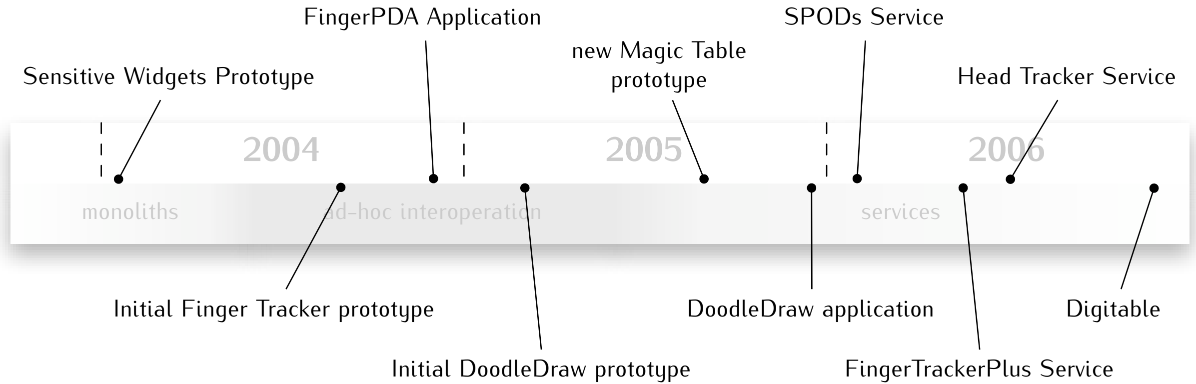


Figure 7.1 . Historique du déploiement de prototypes utilisant *gmlVision*.

systèmes interactifs que nous avons pu réaliser en l'utilisant. Elle montre également la transition, en parallèle avec ces déploiement, depuis une architecture monolithique jusqu'à l'architecture à services que nous proposons dans ce document.

Nous consacrons la première partie de cette section à une description des différents prototypes ainsi que leurs liens avec la conception de la boîte à outils. La seconde partie est un résumé des leçons que nous retenons de ces déploiements. Elle concerne en particulier les points faibles de *gmlVision* et de l'approche sous-jacente.

7.1.1 Historique du déploiement

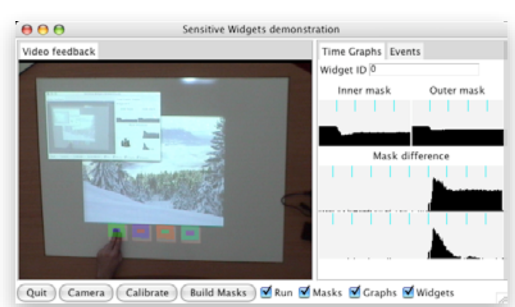
En analysant *a posteriori* les réalisations fondées sur *gmlVision*, on peut distinguer grossièrement trois « ères » qui correspondent à ses évolutions structurelles (voir figure 7.1). Durant l'ère « monolithique » *gmlVision* est une bibliothèque modulaire, mais faiblement structurée, de composants de vision et d'interface assemblés de manière *ad hoc* pour construire des applications. La construction est coûteuses et les applications construites sont de simples preuves de concept. Durant la seconde période, les requis sructurels apparaissent et l'**isolation** et l'**abstraction** prennent le pas sur la performance permise aux monolithes : les prototypes peuvent devenir plus complexes. Enfin, durant la dernière période, la notion de service est établie et les applications sont des fédérations de services perceptifs, de services support, et de services de sortie.

Nous décrivons ici les prototypes réalisés par ordre chronologique.

Dans la suite, nous nommons « proto-service » les services qui sont antérieurs à l'identification et l'utilisation de notre définition d'un service (définition fondée sur les concept d'asynchronisme, abstraction, isolation et contrat, et présentée en 4.3.2 page 81).

7.1.1.1 Prototype Sensitive Widgets (01/2004)

Sensitive widgets est le premier prototype de logiciel perceptif construit en utilisant une version de *gmlVision*. L'application minimale qui l'exploite est *PhotoTouch* : un diaporama tactile muni d'une interface minimale. Cette interface a quatre boutons permettant d'atteindre le début ou la fin du diaporama et de visionner l'image précédente ou suivante (figure 7.2).



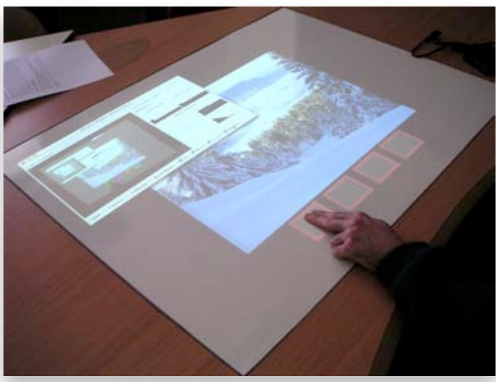


Figure 7.2 . Le premier prototype *Sensitive Widgets*.

À gauche : l'ancêtre du moniteur de service de *gmlVision*. Une interface *ad hoc* permet de configurer et contrôler le proto-service (rangées de boutons et de cases à cocher) et d'inspecter son fonctionnement. Par exemple, le graphique du milieu représente la réponse d'un des « widgets sensibles » en fonction du temps. À droite : une vue de l'application de démonstration (un diaporama tactile). Devant l'utilisateur, sous la photographie courante, la rangée de boutons sensibles permet de naviguer dans le diaporama. Le moniteur est également visible.

Ce proto-service s'adresse aux surfaces augmentées « simples », c'est-à-dire les surfaces où les techniques d'interaction utilisées sont minimales, par une interaction sous forme de pages web. Ces techniques d'interaction ne nécessitent pas de localisation précise de l'agent interactif, ne permettent l'utilisation que d'un agent à la fois, mais peuvent permettre la mobilité : par exemple être projetées sur une surface mobile comme le PDS.

Requis.

Les requis fonctionnels de *Sensitive Widgets* sont de permettre la création de boutons sensibles à l'occlusion par un doigt définis dans les coordonnées de l'interface utilisateur. Des requis non fonctionnels particuliers sont également définis : les conditions d'éclairage ne peuvent être contrôlées, et aucune contrainte n'est placée sur l'interface projetée sur la surface. Le coût en calcul doit être faible afin de permettre l'utilisation de nombreux boutons et de déployer le service sur un ordinateur embarqué (dans une *smart camera* par exemple). Enfin, la latence tolérable est de 1000 ms. à l'initialisation (i.e. lors de la définition d'un bouton par Laurence), et de 250 ms. pour l'interaction (i.e. l'activation d'un bouton).

Ces requis correspondent informellement au besoin que nous avons noté **S14** (c.f. page 49).

Réalisation.

En termes de vision par ordinateur, les widgets tactiles sont les ancètres des *striplets* décrits au chapitre précédent. Il sont composés de deux rectangles concentriques, le rectangle intérieur étant de la taille d'un doigt, dont les variations de luminosité apparente moyenne sont comparées. Ils sont décrits plus en détails dans [Borkowski et al., 2005].

L'architecture de ce prototype est monolithique. Le client dispose d'une classe `SensitiveWidgets` qu'il doit instancier. En entrée du proto-service l'interface proposée inclut des commandes de création et de destruction de widgets et des commandes de réglage de la matrice de calibrage. En sortie, le proto-service génère des commandes d'activation des widgets. En utilisant les notations du chapitre 5, l'interface est :

- ⇠ `<Matrix>` *matrix*
- ⇠ `<Widget>` *id left top right bottom*
- ⇠ `<Destroy>` *id*
- ⇢ `<Click>` *id*

D'un point de vue technique la communication n'est pas événementielle : `<Matrix>`, `<Widget>` et `<Destroy>` correspondent à des appels de méthode, et les `<Click>` sont obtenus par polling d'une autre méthode. Les coordonnées de création sont données dans le repère de l'interface projetée. Le calibrage géométrique doit être effectué à l'extérieur du proto-service par une autre application ou par le client. Le calibrage est ensuite fourni au proto-service *via* la méthode `<Matrix>` ou saisie par l'utilisateur final sur son interface de contrôle.

Le contrôle et la surveillance du service sont permis par une interface graphique *ad hoc*. Elle permet de régler manuellement des paramètres de bas niveau du service (délais de la machine à états, seuil de déclenchement des widgets). Elle affiche en outre la vue de la caméra à laquelle sont superposés des représentations des widgets tactiles et des graphiques permettant de suivre l'évolution de leurs réponses.

Conclusions du déploiement.

Du point de vue fonctionnel ce premier proto-service est un échec. Il est très sensible aux occlusions parasites : de nombreuses fausses alarmes sont déclenchées. Le calibrage doit être mis à jour régulièrement pour une interface mobile en particulier si l'interaction doit être possible pendant le mouvement. Hors, à chaque mise à jour de la matrice de calibrage, le système perceptif doit effectuer une initialisation avec

une latence de l'order de 1000 ms. En outre, le calibrage fourni doit être particulièrement précis (< 1 pixel camera) afin que l'image affichée à l'endroit ou un widget « observe » la scène ne varie pas. À l'utilisation, le calibrage fourni par le client (l'ancètre de `gml.tracker.surface`, c.f. 5.3.2.5 page 109) est trop grossier : sa précision est de l'ordre de 5 pixels caméra.

Du point de vue structurel, la structure choisie est problématique pour interopérer avec le client. D'une part les autres composants du système sont développés avec d'autres boîte à outils monolithiques (en particulier ImaLab). D'autre part les autres composants ont également besoin d'accéder au flux video issu de la caméra. Après des tentatives insatisfaisantes avec *gmlVision*, le développeur des *Sensitive Widgets* choisit de les reproduire au sein de sa propre architecture en produisant un service fonctionellement équivalent. L'application finale est monolithique. Des images de son fonctionnement sont données figure 7.3.

Si ce premier prototype a été un échec, il montre cependant la nécessité d'une formalisation du **contrat** entre le client le le concepteur du service. L'échec de ce proto-service est en effet due à une mauvaise communication du besoin entre Laurence et Patrick. Il fait également émerger le requis d'**isolation** du service perceptif.

7.1.1.2 Prototype initial du Finger Tracker (10/2004)

Dans l'optique de reproduire les techniques d'interaction utilisées sur la *Table Magique* mais au doigt plutôt qu'avec des jetons, nous avons implémenté l'application *Photo Shuffler* (fig. 7.4 page suivante). Celle-ci affiche des photographies numériques qui peuvent être manipulés à un ou deux doigts de la même manière que les *patches* de la *Table* [Bérard, 2003].

Le prototype de service perceptif que nous décrivons ici se place dans le contexte des surfaces augmentées. Il fournit un support générique de l'interaction au doigt nu. Il s'agit donc cette fois de techniques d'interaction pouvant être complexes, par exemple avec plusieurs utilisateurs utilisant chacun plusieurs doigts.

Requis.
Les requis exprimés lors du développement de *Finger Tracker* sont presque ceux de **S10** (c.f. page 46). Les seules différences concernent la précision et la stabilité statique qui sont distinguées : la première est de 5 mm, la seconde de 1 mm. L'objectif est d'être également robuste à un environnement complexe comportant des objets parasites et mobiles.

Réalisation.
Les choix de techniques de vision et l'implémentation de ce service sont, dans les

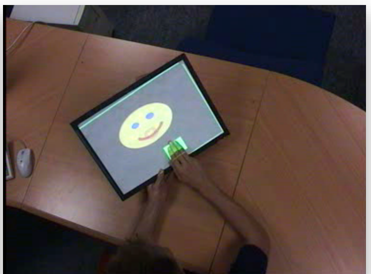


Figure 7.3 . Les *widgets sensibles* utilisés dans le cadre de ContAct.

À gauche, trois utilisateurs autour de l'interface portable ; en bas de l'interface graphique, une rangée de boutons tactiles. Au centre, une vue rapprochée de l'interface. À droite, une interface minimale utilisant un seul widget sensible, qui fonctionne y compris lorsque l'interface est en mouvement et/ou change d'orientation.

grandes lignes, ceux du service `gml.tracker.finger` décrit au chapitre précédent. Il sont détaillés dans le rapport [Letessier, 2003].

La structure du système interactif final n'est plus monolithique. Pour rendre le déploiement plus flexible, le service perceptif et le client applicatif sont isolés dans deux processus communicant par un mécanisme d'ICP sur TCP conçu pour l'occasion. Dans une encapsulation binaire (chaque message est préfixé par 8 octets spécifiant respectivement la taille du paquet et la taille du message dans le paquet), des messages textuels décrivent les événements. Ce mécanisme est un précurseur à BIP, nous le notons BIP/0.0. L'interface en sortie du service consiste en l'émission d'événements de la forme :

> `<Appear>` *id x y timestamp*
> `<Disappear>` *id timestamp*
> `<Motion>` *id x y timestamp*

Le calibrage géométrique est automatique mais long (environ 30 secondes), et doit être effectué à chaque lancement du service. Il requiert l'accès à l'interface graphique par le service et suppose une configuration manuelle : la position et la taille (en coordonnées de l'interface graphique) de la surface à calibrer.

Enfin, le calibrage pour une condition d'éclairage donnée est une opération délicate. Le contraste du projecteur doit être faible, la caméra doit être légèrement sur-exposée afin de ne pas percevoir l'image projetée qui perturberait la détection.

Conclusions du déploiement.

Ce logiciel a été déployé dans notre laboratoire pour la réalisation de tests. Il a ensuite fait l'objet d'une démonstration lors de la conférence UIST 2004 [Letessier et Bérard, 2004] et sert de base au service actuel et a plusieurs prototypes. Sa latence, sa précision, et la performance des utilisateurs (en particulier, vis-à-vis de tâches équivalentes à la souris) ont été évaluées formellement. Nous détaillons ces points dans la section suivante (7.2 page 159).

Il s'agit d'un proto-service : il respecte déjà en partie les contraintes d'isolation et d'abstraction, mais il impose que le service et le client soient colocalisés. La communication avec le client est possible grâce à un canal TCP. Ceci induit une latence non négligeable que nous évaluons entre 5 ms (sur la même machine) et 10 ms (sur le même réseau local). Un moniteur *ad hoc* du fonctionnement du système est fourni mais l'application client n'a aucun moyen d'y accéder.

En termes fonctionnels, un besoin émergent est celui de l'équivalent du clic de la souris pour les interfaces tactiles, notre système ne pouvant détecter le contact. Le

Figure 7.4 . Le premier prototype *Finger Tracker* en fonctionnement.

À gauche : les doigts suivis sont identifiés par des couleurs différentes et des disques de couleur sont projetés au lieu de la détection. À droite : cinq utilisateurs manipules des photographies numériques avec l'application *Photo Shuffler*.

clic est émulé par le client en détectant des pauses dans le déplacement du doigt. Ce besoin est récurrent et devrait être factorisé dans le service : il apparaît également dans d'autres prototypes. Malgré les objectifs initiaux, l'autonomie du système est faible. L'absence de réglages des algorithmes de vision est un progrès sur l'existant. Cependant une supervision humaine est nécessaire au démarrage pour les réglages de la caméra et la calibrage, et en ligne pour « remettre à zéro » le système lorsque l'éclairage a trop varié. Par contre la latence et la précision sont acceptables. Enfin, le contrat restant informel, le logiciel perceptif obtenu est jugé peu robuste par ses utilisateurs : il n'est pas capable de fonctionner dans certaines conditions d'éclairage (s'il n'y a pas de lumière autre que celle projetée, par exemple), et se montre peu robustes aux variations.

7.1.1.3 Application FingerPDA (12/2004)

La présentation d'un ordinateur de poche (PDA) ou d'une de ses applications à une audience nombreuse se heurte aux faibles dimensions de l'appareil. L'objectif de l'application *FingerPDA* (figure 7.5) est de traiter ce problème en utilisant une grande surface projetée en tant que *proxy* tactile de l'écran du PDA.

Cette application utilise le proto-service décrit dans les paragraphes précédents.

Requis.
L'environnement d'usage est très contraint : la lumière est contrôlée, la vue de la

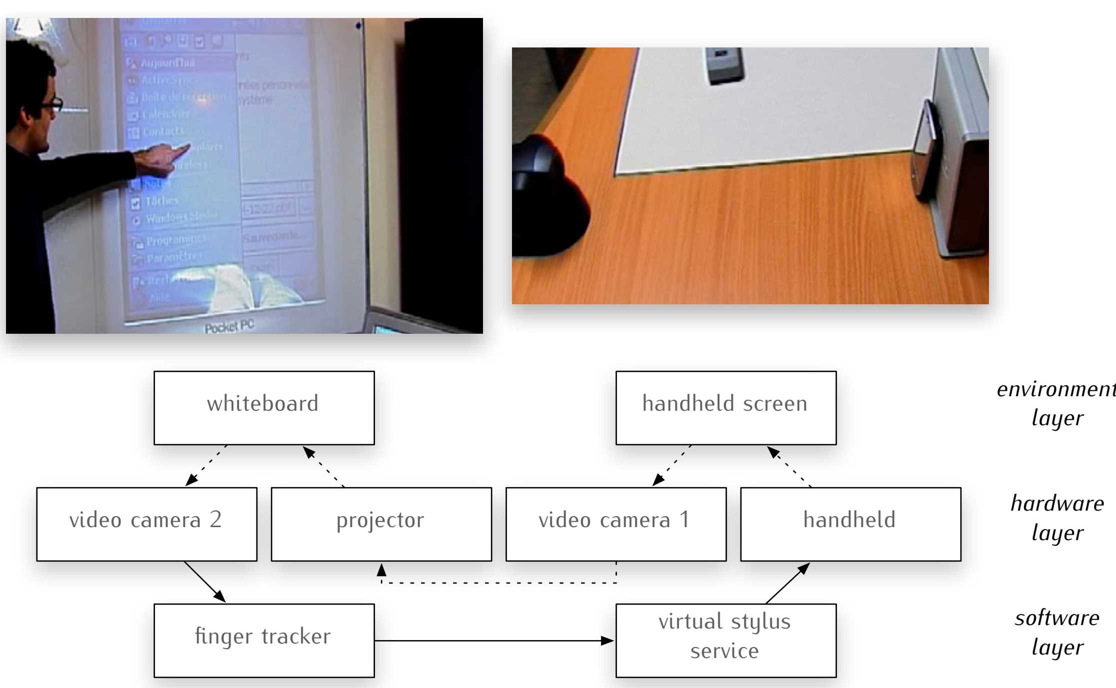


Figure 7.5 . Le prototype *FingerPDA*.

À gauche, l'interface perçue par l'utilisateur : une représentation agrandie et tactile de l'écran d'un PDA. À droite, le reste du dispositif expérimental : une caméra analogique filme l'écran du PDA.
En bas, le schéma-bloc explicatif du flux de l'information dans le système interactif. Les traits en pointillés représente le transport analogique de l'information (par la lumière) ; les traits pleins, par la partie logicielle. En partant de la droite, dans le sens direct : le PDA affiche une interface sur son écran ; la caméra 1 le filme et le videoprojecteur affiche cette interface telle quelle. La caméra 2 observe la projection, et la transmet au service perceptif. Celui-ci interprête le flux video et transmet des événements `<Dwell>`, via `BIP/0.0`, au service « applicatif » qui simule la présence d'un stylet. Ce dernier déclenche la mise à jour de l'interface graphique du PDA.

caméra est exempte de contexte (la scène contient uniquement le tableau blanc). Le point le plus important est que le proto-service doit fournir uniquement les informations sur un seul doigt suivi : l'interface d'un PDA ne gère en effet qu'un point de contrôle.

Réalisation.

Le client étant un novice en vision par ordinateur, la partie perceptive de *FingerPDA* n'affiche plus le moniteur mais une interface minimale. Celle-ci permet le calibrage géométrique manuel (en laissant l'utilisateur cliquer sur les coins de l'écran du PDA dans la vue de la caméra), le démarrage ou l'arrêt du service, et l'affichage du nom de l'hôte et du numéro de port permettant d'exploiter le service.

Le proto-service est modifié pour filtrer les événements avant leur envoi de manière à (*a*) ne produire d'événements que pour un seul doigt et (*b*) tranformer le flux d'événements `Appear`, `Motion`, `Disappear` en un flux d'événements `Dwell` émulant des clis du stylet. Ce filtrage est effectué à l'aide d'une machine à états comparable à celle décrite sur la figure 6.11 page 130.

La communication entre composant sur le PDA et le proto-service ayant lieu sur un réseau 802.11 (sans fil) et non plus sur Ethernet, la latence est augmentée. Le protocole `BIP/0.0` utilisé précedemment est conservé, cependant le transport utilisé est à présent UDP qui fournit une latence inférieure au prix de possibles pertes ou inversions de messages. Certains événements de suivi peuvent être perdus ou arriver dans le désordre. Si le système reçoit un message `<Motion>` plus ancien qu'au moins un message reçu, celui-ci est ignoré car il n'est de toute façons plus interessant. Ceci permet de maintenir une latence comparable à celle obtenue auparavant.

Conclusions.

La supervision humaine au démarrage contribue à diminuer encore l'autonomie : les réglages caméra et le calibrage géométrique manuel doivent être effectués par l'opérateur lors de chaque session d'utilisation. Le besoin de *persistence* des réglages d'une session à l'autre apparaît. La satisfaction de ce besoin permettra de pallier au déficit d'autonomie.

L'utilisation d'UDP, qui ne permet pas de garantir la livraison des messages ni que la connection est maintenue, combinée a la faible lisibilité du protocole, rend cependant difficile le dégoguage des problèmes de connectivité.

En termes de qualité de service, la latence est toujours acceptable. La précision est inutilement élevée (approximativement 0,25 pixel du PDA). Enfin, l'environnement d'usage étant plus contraint, le proto-service (au demeurant identique à celui utilisé auparavant) paraît comparativement plus robuste.

7.1.1.4 Prototype initial de DoodleDraw (02/2005)

L'application *DoodleDraw* est un outil de dessin au doigt sur surface augmentée mobile. Ce projet a été réalisé dans le cadre de cette thèse et de celle de Stanislas Borkowski [Borkowski, 2006]. Il vise un double objectif : d'une part explorer, observer, et évaluer l'interaction avec la surface mobile elle-même (i.e. comment la diriger, comment l'exploiter dans un groupe) ; d'autre part valider le service `gml.tracker.finger` et explorer les requis structurels de *gmlVision*.

Réalisation.

Lors de cette première phase l'application doit utiliser trois services de *gmlVision* : les « widgets sensibles », le suivi de doigts, et l'acquisition video. Il devient rédhibitoire de développer un moniteur pour chaque service et de le faire évoluer en parallèle du développement des services. Un moniteur générique est mis en place conjointement avec l'architecture interne à composants.

Le service `gml.tracker.finger` évolue également : il est capable de fonctionner à partir d'une image en infrarouge proche, ce qui permet de le rendre plus performant (le

flux video source étant 4 fois moins volumineux) et robuste aux variations d'éclairage et aux ombres.

L'intégration de l'application se heurte à plusieurs difficultés. Le passage d'un modèle conceptuel monolithique à un modèle *peer to peer* à services n'est pas trivial : les flux d'information étaient matérialisés par des appels synchrones de méthodes et deviennent des événements asynchrones. Certains services sont implémentés dans un environnement différent, ImaLab, qui par sa nature monolithique se prête mal à l'asynchronisme.

Le protocole de communication `BIP/0.0` est ré-implémenté dans différents environnements ce qui donne lieu à un cycle d'éssais et erreurs de 3 mois-homme environ. Ces cycles aboutissent à la mise en place du protocole bas niveau, puis d'une connection mixte TCP/UDP qui fournit les avantages de la connexion utilisée pour *FingerPDA* et pour le prototype *FingerTracker*. Finalement, nous décidons de normaliser les communications interservices sous la forme d'une spécification qui est la base de `BIP/1.0`.

Conclusions.

Malgré ce progrès le prototype n'aboutit pas. D'une part, certains services externes à *gmlVision*, tel que le detecteur de surface mobile *PDStracker*, ne fournissent pas une qualité de service suffisante. D'autre part, le problème d'accès concurrent au flux video n'est pas résolu. Un besoin émerge : il faut un service `gml.grabber.camera` capable de fournir des images à plusieurs clients, et doté d'une interface facilement accessible, par exemple *via* `BIP`.

L'application est structurellement instable et difficile à déployer. Lors d'échecs ou d'erreur de communication aucun mécanisme de reprise sur panne n'est prévu. Lors du déploiement il est nécessaire de configurer manuellement les numéros de ports des différents canaux de communication et de lancer les services dans un ordre défini. Le requis d'**isolation** apparaît à nouveau. Il implique la possibilité de découvrir les services de manière décentralisée. Un tel mécanisme est incorporé à `BIP`.

7.1.1.5 Prototype de la Table Magique (09/2005)

Nous nous proposons de ré-implémenter un sous-ensemble de la *Table Magique* (fig. 3.9 page 48) en utilisant *gmlVision* : la manipulation des *patches* avec des jetons colorés. Nous ommettons la capture de l'écriture faute de temps pour developper le service perceptif correspondant et de matériel adapté. Notre priorité est de valider l'approche plus que de reproduire le système interactif à l'identique. L'application finale sera en fait *PhotoShuffler*.

Requis.

La *Table* utilise des jetons circulaires colorés pour l'interaction. Nous devons donc construire un service permettant de les suivre avec une latence faible et exacement la même interface que le suivi de doigts. Nous choisissons une implémentation simpliste : une modèle de teinte des jetons dans l'espace *CrCb* (chrominance) est construit à l'initialisation sous forme d'un histogramme avec la coopération de l'utilisateur. La détection des jetons est effectuée par rétropropagation de l'histogramme dans chaque image perçue. La suite du mécanisme est similaire à celle de `tracker.finger` : la carte obtenue est seuillée automatiquement, une analyse en composante connexe permet d'isoler les jetons. Les composantes de dimensions anormales sont rejetées. Les centres des composantes restantes sont injectées dans le composant d'association de `tracker.finger`.

La possibilité de suivre des jetons de plusieurs couleurs est souhaitée. Bien qu'il soit possible d'implémenter cette fonctionnalité dans le même service, nous choisissons de conserver une interface simple, car il suffit de déployer plusieurs services de suivi de jetons utilisant des modèles de couleur différents pour ce faire (en supposant que le coût du middleware est négligeable).

Réalisation.

La figure 7.6 synthétise l'architecture de ce prototype. Si l'implémentation du service perceptif est simpliste, l'architecture dans toute ses facettes est mise à l'épreuve par l'intégration des différents services.

Le déploiement de ce système coïncide avec l'introduction de la spécification BIP/1.0 presque sous sa forme actuelle (à des conventions de nommage près, et sans spécification du protocole de contrôle sur XML). Par ailleurs, le moniteur générique déployé pour *DoodleDraw* fonctionne sans modifications mais impose une surcharge élevée (10 à 30% du temps de calcul) ce qui augmente la latence moyenne du système perceptif jusqu'à 120 ms.

Le calibrage automatique est assuré par un service distinct ayant un protocole normalisé (celui décrit en 5.3.4.1 page 111). Il collabore avec un service d'affichage lui aussi indépendant. L'affichage est prévu pour être un processus serveur toujours disponible sur la machine hôte, qui doit être déployé par Caroline.

L'état des services (leurs paramètres et en particulier les données de calibrage) est persistent au travers de plusieurs sessions d'utilisation, ce qui rend l'expérimentation plus aisée.

Le lancement et la connexion des différents services est effectué par un script « d'emballage » qui est spécifique à la *Table Magique* et à la situation dans laquelle il a été testé (i.e. aucun autre service ne doit être présent sur la machine de test).

Conclusions.

Le moniteur générique de services est invasif et perd son intérêt : s'il impacte la latence et la fréquence de fonctionnement, certains phénomènes ne sont plus visibles. Le monitoring doit donc être plus dissocié du service et ne pas « voler » de temps de calcul au service perceptif.

La mise en œuvre de *gmlVision* sous la forme d'un monolithe ou d'une bibliothèque « classique » aurait évité certaines des difficultés de déploiement rencontrés dans *DoodleDraw*, mais aurait rendu difficile la réutilisation de ses composants. En outre la stabilisation du prototype aurait été plus longue : l'architecture faiblement couplée de la boîte à outils permet d'effectuer aisément des tests unitaires des service. Cette architecture simplifie également les tests d'intégration : dans *gmlVision* n'importe quel service peut être stoppé ou tomber en panne puis être relancé indépendemment des autres.

Cependant, du point de vue de Laurence, un monolithe est plus utilisable qu'une fédération de services : les services doivent être instanciés indépendemment ce qui complique le lancement de l'application. La solution palliative, un script qui lance les services dans l'ordre, n'est pas satisfaisante : elle n'est pas générale (le script est propre à chaque déploiement) et impose un effort de développement supplémentaire à Laurence (qui doit écrire ce script). Il s'agit cependant du premier exemple d'empaquetage (ou surperviseur) de services dans *gmlVision*.

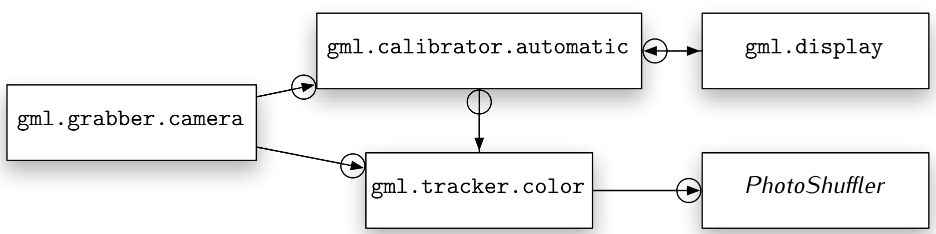


Figure 7.6 . Architecture concrètre du prototype de *Table Magique*

Les flêches indiquent la direction des flux d'évèments ; les cercles désignent le service qui initie la connection.

Le problème qui apparaît peut être formulé ainsi : comment libérer Laurence de la charge d'instancier les services et d'initier les connections ? La solution envisagée, simple incrément du script d'instantiation, est de fournir un exécutable qui « empaquète » plusieurs services transformant ainsi la fédération de services en monolithe. Cependant, la solution qui apparaît comme la plus satisfaisante est de fournir un service « chapeau » qui matérialise le couplage entre services (ce service deviendra `gml.surface`) et se charge de l'instanctiation et de la connection entre services.

7.1.1.6 Application DoodleDraw (12/2005)

Un deuxième prototype de *DoodleDraw* a put être réalisé et déployé une fois les outils logiciels emballés sous forme de services `BIP/1.0` (suivi de surface, contrôle du vidéoprojecteur orientable, service d'accès et partage de caméra video). Cette application a été conçue et dévelopée en collaboration avec Stanislas Borkowski, Dominique Vaufreydaz, et Jérôme Maisonnasse.

Requis.
En dehors des requis fonctionnels et structurels évoqués dans la description du premier prototype, *DoodleDraw* possède une particularité contraignante : elle permet plusieurs modalités pour le contrôle de la position de l'interface graphique projetée. Ces modalités sont décrites et étudiées en détail dans [Borkowski et al., 2006]. En résumant, ces quatres modalités sont :

- aucun contrôle : l'interface est statique (cas de référence) ;
- contrôle explicite : l'interface est munie d'un bouton par utilisateur. La sélection d'un bouton déplace l'interface vers cet utilisateur (par pilotage du videoprojecteur orientable) ;
- contrôle tangible : l'interface est en permanence projetée sur un PDS qui peut être déplacé par les utilisateurs ;

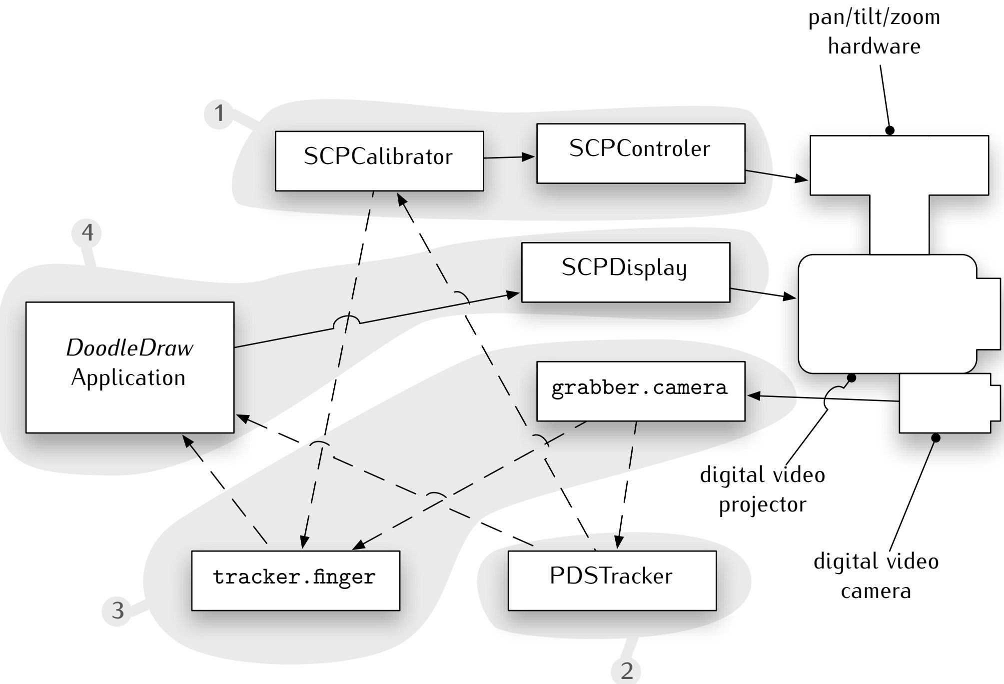


Figure 7.7 . Architecture du prototype *DoodleDraw* dans l'une des quatre conditions expérimentales.

Chaque rectangle représente un service. Les flêches discontinues représentent les communications utilisant `BIP/1.0` ; les flêches continues, les communications *ad hoc* entre services.
Les contours grisés représentent l'exécutable qui instancie les services inclus dans le contour. Les chiffres qui leur sont attachés désignent l'ordre (fixe) de lancement.

– contrôle implicite : un service d'audition artificielle analyse les tours de parole des utilisateurs, infère un « leader », et lui confie l'interface.

En termes structurels ceci signifie que *DoodleDraw* utilise des services perceptifs différents. Le schéma d'intégration (figure 7.7 page précédente) correspond à la troisième situation. Le service *PDSTracker* (suivi du PDS) sera remplacé par d'autres services dans les autres situations.

Il n'est pas réaliste de déployer une application pour chacune des conditions expérimentales car cela demanderait un effort de développement trop important. Le choix d'une architecture à services ainsi que le requis d'**isolation** sous-jacent s'en trouvent renforcés.

Réalisation.

Le service `tracker.finger`, dont le contrat ne prévoit pas l'utilisation pour une vue ou une surface d'interaction mobile, doit être adapté. L'interface de contrôle permet la modification des paramètres de calibrage. Cependant les intégrateurs de *DoodleDraw* choisissent de « bricoler » le service en permettant à un client d'injecter une nouvelle matrice de calibrage par le canal *reset* (c.f. page 105) car les implémentations existantes de `BIP/1.0` ne peuvent utiliser le canal de contrôle de manière simple et fiable. Dans les deux cas, ceci entraîne l'arrêt de la perception et l'émission d'un événement `<ResetQuery>`.

Le moniteur est desactivé par défaut. L'overhead de surveillance des services n'a lieu que si le moniteur est affiché. Chaque événement `<Monitor>` n'est traité que lorsque du temps de calcul est disponible. L'augmentation de latence induite par le moniteur est minimale (environ 5 ms). En contrepartie la fréquence de rafraîchissement du moniteur est souvent inférieure à celle du flux video (environ 20 Hz contre 30 Hz).

Enfin, basculer de l'une à l'autre des quatres conditions expérimentales n'est pas possible dynamiquement. La quasi-totalité des exécutables empaquetant des services doivent être relancés lorsque la condition changée. Bien que ceci aurait pu être dynamique (i.e. sans relancer de programmes), les paquetages de services ne sont pas prévus pour les re-connections. Les auteurs ont envisagé de construire une application graphique « superviseur » capable de modifier l'interconnection des services selon la condition et d'instancier les services manquants. Ce projet n'est pas concrétisé car il demande un effort de developpement considéré superflu.

La figure 7.8 présente l'application en cours d'utilisation. Elle a été utilisée par environ 250 utilisateurs durant son déploiement. Nous reviendrons sur son évaluation dans les sections suivantes.

Figure 7.8 . L'application *DoodleDraw* en cours d'utilisation.

À gauche, le matériel utilisé. Un video-projecteur, une caméra infrarouge et une lampe infrarouge sont montés sur une structure pilotable selon deux degrés de liberté. Leurs axes optiques sont quasi-parallèles. Elle permet de créer des surfaces interactives dans un hémisphère.
Au centre et à droite, deux vues d'utilisateurs effectuant des tâches de dessin libre avec l'application. La première montre l'interface portable PDS. Dans la seconde, l'interface est simplement projetée sur la table.

Conclusions.

DoodleDraw a été le support d'une expérimentation d'IHM assez complexe impliquant de nombreux utilisateurs, plusieurs conditions expérimentales, et plusieurs modalités et techniques d'interaction. Nous pouvons donc affirmer que son déploiement est un succès.

L'architecture s'est montrées suffisemment adaptée pour intégrer un ensemble de 7 composants (6 services et l'application interactive). Les spécifications d'un service, utilisant la *lingua franca* de `BIP/1.0` et d'un format d'événements standardisé, ont permis à un groupe de developpeurs d'intégrer se prototypes en se souciant plus de leur propre composant que de l'interopération.

Cependant certains problèmes de déploiement persistent : pour faire fonctionner l'application interactive Caroline doit lancer plusieurs exécutables sur plusieurs machines dans un ordre donné. Ceci est répété à chaque changement de condition expérimentale.

7.1.1.7 Service SPODs (01/2006)

Depuis l'échec relatif des *Sensitive Widgets* de nouveau algorithmes de vision par ordinateur ont été mis au point pour permettre de satisfaire les besoins d'interaction tactile dans un environnement moins contraint que celui imposé par le contrat de `gml.tracker.finger`. Il s'agit des striplets et SPODs introduits au chapitre précédent (6.2.2 page 132).

Réalisation.

Nous implémentons une application simple pour expérimenter avec ce service perceptif. Appelée *TouchCalc*, c'est une calculette tactile (figure 7.9).

Lors du développement les deux composants perceptifs, striplets et couche d'abstraction VEIL (c.f. figure 6.13 page 132), sont développés par deux personnes. Le premier dans le contexte de *gmlVision* et le second en utilisant le middleware *OMiSCID* pour l'interopération. L'environnement logiciel est donc hétérogène. La communication entre composants se fait *via* la couche intermédiaire de `BIP/1.0` : les services sont découverts dynamiquement et désignés par leur classe et leur nom plutôt que

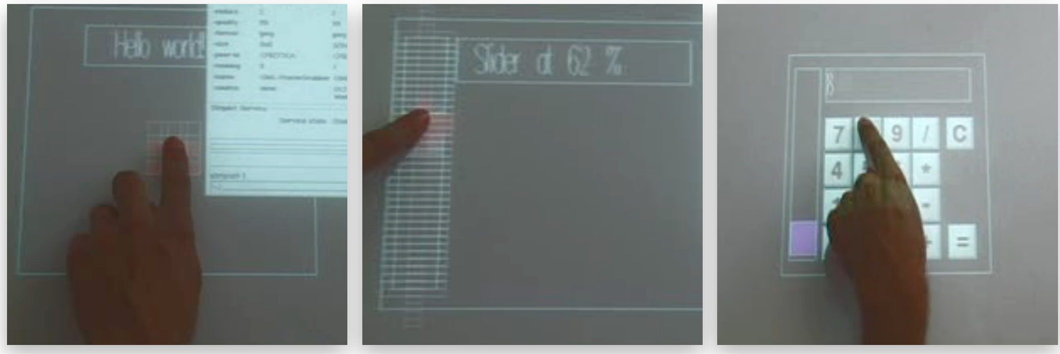


Figure 7.9 . *TouchCalc*, un prototype qui utilise les striplets et SPODs.

À gauche, une fédération de 6 striplets permet de construire un bouton tactile. Un ensemble d'heuristiques sur les occlusions des striplets qui composent le bouton détermine son activation éventuelle.
Au centre, une fédération plus complexe permet de construire un *slider* (barre de défilement). Le barycentre des striplets occultés fournit la position du curseur.
À droite, l'applications graphique démontrant l'utilisation du service SPODs. Une calculette tactile, *TouchCalc*, est munie d'un ensemble de boutons pour la saisie des chiffres et des opérations et d'une barre de défilement pour l'historique.

par une adresse IP et un numéro de port. Par contre la robustesse des connexions n'est pas assurée : si l'un des services échoue il est nécessaire de relancer l'ensemble.

Le besoin d'un service d'acquisition video qui peut être partagé entre clients est toujours impératif. `gml.grabber.camera`, déjà déployé pour *DoodleDraw*, est ici finalisé. Un client a un effort minimal à fournir pour accéder au flux video : établir une connexion `BIP` sur TCP, puis recevoir les images avec `BIP` sur SHM (mémoire partagée locale) ou sur TCP.

Conclusions.

L'architecure orientée services, fondée sur l'utilisation de `BIP`, atteint ses objectifs de flexibilité sans pénaliser la latence totale du système interactif. Elle est en particulier bien adaptée aux machines parallèles (i.e. la plupart des machines modernes) : les traitements indépendants peuvent être effectués simultanéments. Ici, les quatre traitements sont le décodage du flux video, l'analyse des striplets, l'intégration des SPODs, et la gestion de l'interface utilisateur. Ce parallèlisme au niveau service est assimilable à du *software pipelining* [Allan et al., 1995]. Il réduit (marginalement) la latence et augmente la fréquence de l'ensemble du système interactif.

Le déploiement de *TouchCalc* met en évidence la frontière floue entre service et composants. D'après la méthode de conception présenté au chapitre 5 `gml.tracker.widgets` devrait être un seul service. Les intégrateurs (Patrick) ont choisi d'en faire deux services parce que c'est plus facile pour eux.

7.1.1.8 Service *FingerTracker+* (05/2006)

Jean-François Vandamme (Université de Liège), un utilisateur de *gmlVision*, a étendu le service `gml.tracker.finger` pour permettre le suivi des mains en plus de celui des doigts. Le nouveau service résultant, *FingerTracker+*, est également plus robuste : il produit moins de fausses alarmes lorsque les conditions d'éclairage sont à la limite du contrat.

Ce service démontre l'extensibilité des couches de « bas niveau » (en langages C et Lg de la bibliothèque). Le développeur a joué le rôle de Stanislas. Il était familiarisé à la vision par ordinateur (quoique non expert), mais pas avec *gmlVision*. Avec la supervision du concepteur de *gmlVision*, il est parvenu à un prototype de service utilisable en 3 mois.

7.1.1.9 Application *HeadMouse* (06/2006)

Cette application utilise le service `gml.tracker.face` (section 6.2.3 page 136) pour permettre à l'utilisateur de contrôler le pointeur d'une interface graphique ou d'émuler la molette d'une souris par des mouvements de la tête, reproduisant ainsi la fonctionnalité de la *Perceptual Window* [Bérard, 1999a].

Deux services supplémentaires sont développés pour construire l'application. `gml.filter.mouse` transforme les évènement de `gml.tracker.face` en de « faux » événements souris au système d'exploitation. Le pointeur est asservi en position : tout mouvement du visage déclenche un mouvement du pointeur, dans la même direction, avec un gain fixe. `gml.grabber.keyboard` intercepte l'appui d'une touche du clavier pour permettre le *clutching* : le contrôle du pointeur n'a lieu que lorsque la touche est maintenue appuyée par l'utilisateur.

Cette démonstration est l'occasion d'expérimenter avec une application particulière, n'ayant pas d'interface graphique : elle sert simplement de service perceptif pour un système d'exploitation non prévu à cet effet. Elle montre que *gmlVision* peut permettre d'augmenter des applications existantes avec plusieurs modalités de perception et d'interaction.

7.1.1.10 Digitable (12/2006)

La *Digitable* [Digitable, 2007] est un système expérimental destiné à améliorer les possibilités d'interaction sur surface interactives et à expérimenter sur de nouvelles formes d'interaction et de nouvelles applications. Ce système est conçu dans le cadre du projet ANR/RNTL DIGITABLE. L'auteur n'a pas participé à son développement ; par contre certains autres membres de l'équipe IIHM y ont participé. La Digitable utilise plusieurs modalités de perception, dont un suivi de doigts fondé sur le service `tracker.finger` de *gmlVision*.

Les concepteurs de la Digitable ont adapté *gmlVision* pour remplacer le protocole `BIP/1.0` par un protocole *ad hoc* sur UDP : ils n'ont pas jugé interessant l'apport du protocole, ou de l'architecture à services, pour leur projet. Le composant `ServiceChannel` représentant le canal *tracking-events* a été remplacé par un composant écrit pour l'occasion. Le développeur de *Digitable* chargé de l'intégration n'a pas souhaité investir du temps dans l'apprentissage de `BIP/1.0` et le développement d'un client ; il préfère se servir de ses propres connaissances et outils sur UDP.

Bien que ceci constitue un échec pour `BIP`, nous pouvons en tirer deux conclusions positives pour *gmlVision* :

– la structure à services et composants a permi au développeur (jouant le rôle de Patrick) d'intégrer un service de notre boîte à outils à un système interactif n'utilisant pas *gmlVision* par ailleurs, au prix d'une intervention dans le code de la boîte à outils ;
– le service `tracker.finger` est bien ciblé, et suffisement utilisable, puisque les concepteurs de *Digitable* ont choisi de l'exploiter.

7.1.2 Conclusions

Les différentes applications interactives construites pendant l'élaboration de cette thèse valident notre approche : elles ont permis à des personnes prenant les rôles de Stanislas, Patrick, Laurence, et Caroline de l'utiliser pour respectivement créer des composants de vision, des services perceptifs, des applications interactives, et utiliser ces applications. *gmlVision* est utilisée pour les services qu'elle offre, non pas simplement en tant qu'expérimentation pour tester la boîte à outils.

Cette expérience nous permet également de mettre en avant certaines faiblesses de la bibliothèque *gmlVision*.

7.1.2.1 Services critiques manquants

Le défaut d'autonomie commun à presque tous les services que nous avons déployés est lié aux capteurs utilisés. Il est généralement nécessaire de procéder manuellement à leur réglage avant une session d'utilisation d'un système interactif, ou (au moins) lors de leur installation.

L'aspect d'une scène est lié à un ensemble de paramètres d'une caméra : courbe de gain, niveaux du blanc et du noir, ouverture de l'iris, temps d'exposition, balance des blancs, etc. Il est nécessaire de régler les paramètres pour obtenir une « bonne » image, c'est à dire une image qui contient le maximum d'information. De plus, la plupart des algorithmes de vision ne peuvent fonctionner que si cet aspect est constant, c'est-à-dire si les paramètres ne varient pas au cours du temps.

Les caméras numériques sont capables de régler ces paramètres automatiquement et dynamiquement de manière à produire une image satisfaisante pour l'œil humain. Malheureusement ces réglages automatiques ne sont pas satisfaisants pour nous :

– ils sont variables. Ces variations ne sont par forcément visibles pour l'œil humain mais elles sont critiques pour les algorithmes de vision.
– ils sont mauvais. Une image « visuellement satisfaisante » n'est pas nécessairement celle qui contient le plus d'information ou l'information la moins déformée pour la perception.

Il est donc souhaitable de reproduire les mécanisme de gain/exposition/iris automatique des caméras sous forme logicielle. Une telle fonctionnalité pourrait être implémentée dans un service `calibrator.lighting`. Elle pourrait être réalisée en calculant la quantité d'information présente dans une image (estimée par exemple en mesurant l'entropie de son histogramme de luminance) et en déterminant l'ensemble de paramètres qui la maximise. Remarquons que certains paramètres de caméras ne sont pas toujours modifiables de manière logicielle. Par exemple l'ouverture de l'iris est souvent mécanique. Un tel service devrait dans ce cas coopérer avec l'utilisateur final (Caroline).

7.1.2.2 Conception d'applications

Pour l'instant la construction de fédérations de services est hasardeuse : nous manquons de règles de conception. En particulier, hors des cas triviaux (un service, une application), nous n'avons pas de réponse aux questions suivantes : comment « bien » empaqueter les services, c'est-à-dire, de manière utilisable pour Laurence ? Qui doit instancier les services ? Qui est chargé de les connecter, les maintenir connectés, et assurer le dynamisme de la structure ? L'architecture à services est par nature *peer-to-peer* et antagoniste avec le modèle linéaire et monolithique des applications traditionnelles qui ne peut donc nous éclairer. Il sera nécessaire d'explorer le domaine plus avant pour identifier des patrons de conception (*design patterns*).

Une solution est proposée et partiellement implémentée par les développeurs de *jOMiSCID* [Reignier et al., 2006] : chaque machine de l'environnement, dotée de capacités de perception, d'action, ou de calcul, exécute en permanence un « lanceur » de services. Un outil graphique permet de bâtir des fédérations de services. Cette solution a l'inconvénient d'être particulièrement lourde : elle élève considérablement le seuil fonctionnel pour Laurence et Caroline. En effet, on impose alors à Laurence de manipuler un mécanisme de découverte et lancement de services (qui était optionnel dans `BIP`) ; et Caroline doit installer le lanceur de services sur chaque machine.

7.1.2.3 Intégration de services

Notre expérience indique que l'intégration de services devrait être rendue plus aisée pour Patrick.

Le framework fourni par la classe `Service` de *gmlVision* pour l'empaquetage de services pourrait être complété. Une grande partie de l'effort de développement de Patrick est consacré à l'écriture de l'application principale qui empaquète un ou plusieurs services. Cet effort concerne en particulier de son interface graphique. En s'appuyant sur les techniques de génération d'interface graphique (IG) mises en place pour le composant `ServiceMonitor`, il est possible d'ajouter à `Service` la possibilité de construire le composant d'IG qui permet à Caroline de le configurer (c.f. exemple figure 6.12 page 131).

Le service `gml.surface`, qui n'est pas encore finalisé, est critique pour le déploiement En effet, Il permet de s'affranchir de plusieurs limitations évoquées ci-dessus. Il s'agit de l'implémentation d'un patron de conception observé lors de l'expérimentation : le couplage entre un service perceptif, un service de capture video, et un service d'affichage. Ce couplage est identifiable à la notion de surface interactive. Cette réflexion peut potentiellement être généralisée à des lieux d'interaction plus complexes (bureau augmenté, bâtiment augmenté) ou moins complexe (surface tactile sans affichage) comme dans [Fails et Olsen Jr., 2002]) : existe-il d'autres patrons ? Comment les implémenter de manière générale ?

L'assemblage des composants au sein d'un service est rapide mais devient peu lisible, et son comportement peu observable, lorsque le nombre de composants augmente. Il sera donc nécessaire de fournir des outils permettant de simplifier l'assemblage et l'inspection de la coopération entre composants. Il serait sans doute possible d'adapter un

outil graphique existant tel que *Quartz Composer* d'Apple Inc. ou l'outil `gst-editor` de *gstreamer*.

7.1.2.4 Monitoring

Le moniteur fourni à Stanislas et Patrick permet d'inspecter le fonctionnement des services en temps réel. C'est un service très apprécié, mais il présente certaines limites qu'il serait intéressant de combler.

- la surveillance n'est possible que localement (l'IG du moniteur est incluse dans celle du service). Il faudrait permettre la surveillance de services isolés (dans plusieurs programmes, voire sur plusieurs machines) depuis une IG unique.
- le moniteur ne fournit pas d'informations sur les échanges d'événements entre composants ou entre services. Un affichage (graphique) des flux d'évènements permettrait de diagnostiquer des problèmes à l'intégration.
- il ne permet pas de conserver d'historique du fonctionnement qui permettrait d'effectuer des analyses *post mortem* des services. Un journal des connections et des messages échangés pourrait être implémenté.

Il paraît donc intéressant de concevoir un moniteur plus général capable de surveillance « à distance », et de fournir des informations à plusieurs échelles : composant, service, et fédération.

7.2 Évaluation individuelle des services

Il peut paraître surprenant d'évaluer individuellement les services de *gmlVision* : notre objectif est de réaliser une boîte à outils utilisable pour Laurence, et pas (directement) pour Caroline. Cependant, l'utilisabilité des services du point de vue de Caroline est une condition nécessaire à l'utilité de la boîte à outils pour Laurence. Laurence ne peut en effet construire des systèmes interactifs utilisables par Caroline qu'à la condition que sur le « fond », *gmlVision* soit utilisable. Réciproquement, si les services ne sont pas utilisables par Caroline, alors Laurence ne pourra pas construire de système interactif interessant (car elle n'est pas experte en vision) et elle se détournera de *gmlVision*. Il est donc important de tester l'utilisabilité des services du point de vue de Caroline comme défini en introduciton de ce chapitre. L'évaluation individuelle des services participe à la **validation** de la boîte à outils.

Dans le contexte de cette thèse, la méthodologie pour évaluer les services fournis par *gmlVision* est simple : il s'agit de vérifier qu'ils remplissent leur contrat (c.f. 4.3.2.4 page 83). L'hypothèse sous-jacente est que, si le service remplit son contrat, il est utilisable du point de vue de Caroline. En d'autres termes, il s'agit de construire des expériences qui permettent la mesure de la qualité de service fournie, suivant les différents axes, dans tout le « volume » d'environnement spécifié par le contrat.

Cependant, les techniques mises en œuvre pour mesurer la qualité de service peuvent être complexes. Mesurer la latence d'un service perceptif, par exemple, est une tâche lourde qui implique la mise en place d'un appareillage spécifique ainsi que la capture et l'étiquetage manuel d'un flux video [Letessier et Bérard, 2004]. Imaginer la reproduire pour un nombre statistiquement pertinent de conditions expérimentales est donc irréaliste.

Il est également possible d'évaluer directement l'utilisabilité des services du point de vue de Caroline. Cette évaluation se fait lors d'expérimentations utilisateur classiques où un ensemble de sujets effectuent une tâche simple. L'évaluation peut par exemple permettre de comparer les performances des sujets lorsqu'ils utilisent le service perceptif et lorsqu'ils utilisent un technique d'interaction standard. Les tâches simples doivent être choisies de façon à être représentatives : l'évaluation doit pouvoir être généralisée à des tâches réelles plus complexe.

Comme ce travail doctoral a une vocation exploratoire, nous avons pu évaluer formellement uniquement le service `gml.tracker.finger` et, dans une moindre mesure, le service `gml.tracker.widgets`. Ces évalutations ont fait l'object de publications [Letessier et Bérard, 2004, Borkowski, 2006]. Nous en fournissons ici une version plus détaillée.

Nous terminons cette section par une comparaison des services fournis dans *gmlVision* avec d'autres systèmes existants.

7.2.1 gml.tracker.finger

Ce service a subi un test formel d'utilisabilité hors du contexte d'une application particulière, et des évaluations quantitatives et qualitatives dans deux contextes : l'application *PhotoShuffler*, et l'application *DoodleDraw*.

7.2.1.1 Test d'utilisabilité

Nous avons réalisé une expérience destinée à évaluer l'utilisabilité d'une surface tactile fondée sur l'utilisation de `gml.tracker.finger` vis-à-vis de tâches équivalentes réalisées à la souris.

Un groupe d'utilisateurs est chargé d'effectuer une successions de tâches de Fitts (acquisition de cibles de taille variable et à distance variable), dans trois conditions différentes, représentées sur les trois images. Nous avons demandé à 12 sujets de sélectionner 18 cibles circulaires de rayon variables et placés à des distances variables de positions de départs fixes. Tous les sujets sont experts en manipulation à la souris, mais n'ont jamais utilisé un système de suivi de doigts. Chaque sujet a répété l'expérience dans trois conditions (figure 7.10), l'ordre de passage des conditions est balancé entre les sujets.

Dans la première condition, le sujet active une cible en y plaçant son index, et en l'y maintenant pendant un temps de pause (*dwelling*) fixé à 300 ms. Dans la seconde condition (condition de référence), il clique à l'intérieur de la cible en utilisant la souris. Dans la dernière condition, un pointeur est projeté à proximité du doigt détecté, et toujours à la même distance. Le sujet place le pointeur dans la cible et l'y maintient pendant 300 ms. Nous mesurons les temps de réalisation pour chaque tâche élémentaire (figure 7.12 page 162).

La conclusion de cette expérience est que `gml.tracker.finger` permet la manipulation au doigt avec une efficacité proche de celle de la souris. Avec le système actuel,

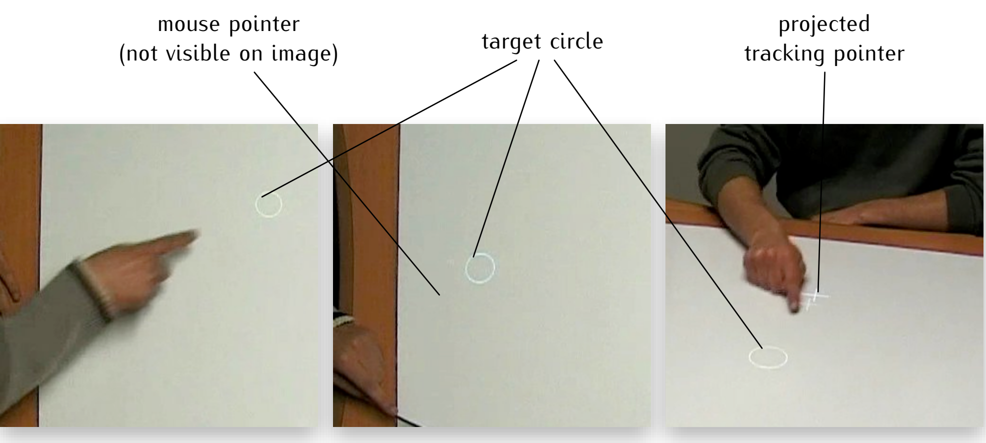


Figure 7.10 . Conditions du test d'utilisabilité de `gml.tracker.finger`.

À gauche, la condition *Finger*. Au centre, la condition *Mouse* (condition de référence). À droite, la contion *Pointer*.

incapable de détecter le contact du doigt sur la surface, l'efficacité sera diminuée par le temps de pause de 300 ms nécessaire pour valider la sélection. Cependant, nous pensons que ceci est contrebalancé par (*a*) la possibilité d'interaction à deux mains et (ou) à plusieurs doigts, et (*b*) la possibilité de créer des techniques d'interaction pour la sélection ne nécessitant pas de temps de pause, par exemple utilisant la technique de *crossing* [Moran et al., 1997, Accot et Zhai, 2002] plus adaptée aux interfaces tactiles (au stylet, ou au doigt).

Il est intéressant de noter la différence de performance (significative dans notre expérience) entre les conditions *Finger* et *Pointer*. Ces deux conditions utilisent le même dispositif de pointage : le service `gml.tracker.finger`, elles sont donc sujettes à une latence identique. Cependant, cette latence est perceptible dans la condition *Pointer* mais pas dans la condition *Finger*. En condition *Finger*, le retour d'information utilisé par les sujets dans leur tâche d'acquisition est la position de leur doigt. Ce retour a donc une latence quasi nulle (c'est le temps de propagation de la lumière sur une courte distance). C'est un des principaux bénéfice d'une manipulation réellement directe. Cependant, cette technique est applicable uniquement lorsque les tailles des cibles sont de l'ordre de la taille d'un doigt. Pour des cibles plus petites il est indispensable de projeter un pointeur. Il est donc nécessaire de travailler à la réduction de la latence de notre service pour améliorer les performances d'acquisition sur de petites cibles.

Notons également que les régressions linéaires illustrées sur la figure 7.12 indiquent que le pointage au doigt nu semble plus performant que le pointage à la souris pour des tâches difficiles. Notre interprétation est la suivante : la souris a l'avantage d'un gain en déplacement supérieur à 1 : un petit mouvement de la main (pour déplacer la souris) permet un grand mouvement du pointeur. Le gain dans la condition *Finger*

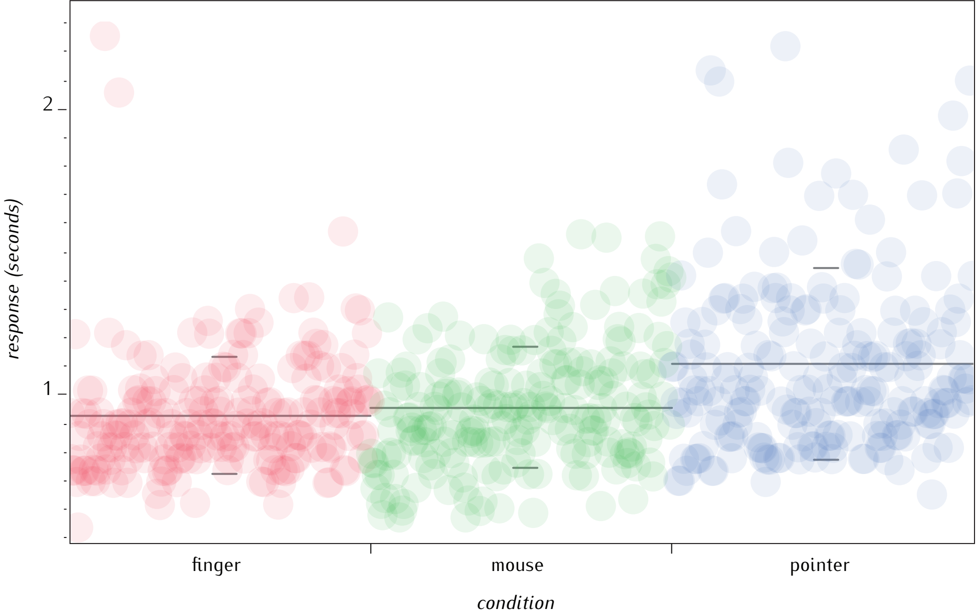


Figure 7.11 . Test d'utilisabilité de `gml.tracker.finger`.

Ce graphique représente la densité des temps de réalisation des tâches élémentaires pour chacune des trois conditions. Chaque point des nuages correspond à une tâche réalisée par un utilisateur. Les barres horizontales représentent la moyenne et l'écart-type des temps de réponse pour chaque condition. Nous avons soustrait le temps de pause de 300 ms aux conditions *Finger* et *Pointer* puisqu'il ne participe pas au temps d'acquisition de la cible.
La moyenne des temps d'acquisition de cible est de 951 ms (dev. 210 ms) dans la condition *Mouse*, 1105 ms (dev. 337 ms) dans la condition *Pointer*, et 923 ms (dev. 203 ms) pour *Finger*. Les différences entre *Pointer* et les deux autres conditions sont significatives ($p < 0{,}0001$), mais pas entre *Finger* et *Mouse* ($p = 0{,}27$).

est strictement 1. Cependant, cet avantage ne joue plus lorsque l'indice de difficulté augmente car ceci implique un accroissement du nombre de cycle perception, cognition, action. Or seuls les tout premiers cycles correspondent à des déplacements de grande ampleur où un gain fort est bénéfique.

Lors du test d'utilisabilité, nous avons observé qu'en moyenne la performance des sujets augmente plus rapidement au cours de l'expérience pour les conditions *Finger* et *Pointer* que pour *Mouse* (figure 7.13 ci-contre). En fin d'expérience, les sujets sont d'ailleurs aussi performants pour *Finger* que pour *Mouse*. Nous en concluons que l'effet d'apprentissage de la tâche est plus marqué pour *Finger*. Nous attribuons ce phénomène au fait que tout nos utilisateurs sont des experts de la condition *Mouse* alors qu'ils n'ont en générale aucune expérience, ou une faible expérience, de la condition *Finger*. Bien sur, toute personne devient experte du pointage au doigt à un jeune age. Cependant, il s'agit du pointage d'objets physiques. Dans le cas du pointage d'objets numériques projetés, il semble qu'un certain temps d'apprentissage est nécessaire pour prendre confiance dans l'efficacité du geste.

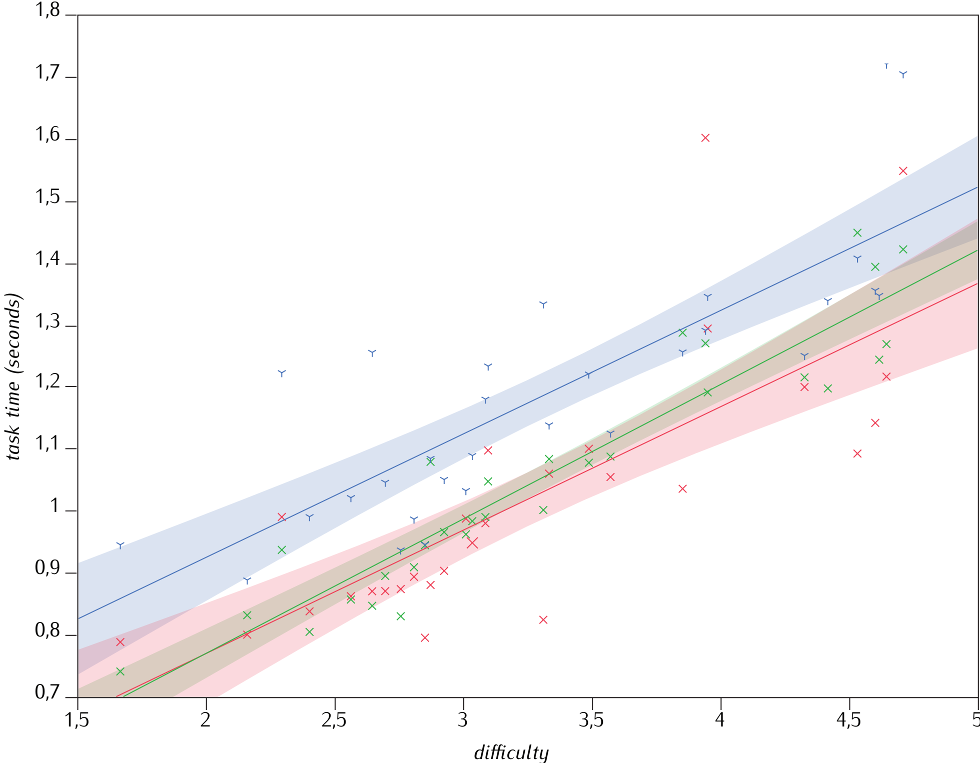


Figure 7.12 . Test d'utilisabilité de `gml.tracker.finger`.

Ce graphique synthétise les temps de réponse des utilisateurs en fonction de la difficulté de la tâche, pour chacune des trois conditions. Le code couleur est repris à la figure 7.11 page précédente.
Selon les abscisses, les indices de difficultés (sans unité), calculés selon la formule de Fitts : $d = log(a/b+1)$, où *a* est la distance de la cible de la cible, *b* son diamètre, et *d* l'indice de difficulté.
Selon les ordonnées, les temps d'accomplissement (en secondes), moyennés sur l'ensemble des utilisateure. La moyenne tronquée à 15% est utilisée pour éliminer les erreurs de mesure grossières observées lors de l'expérience. Nous avons ici aussi soustrait le temps de pause de 300 ms aux conditions *Finger* et *Pointer*.
Comme la loi de Fitts indique que les indices de difficulté et les temps d'accomplissement sont liés linéairement, le graphique inclut les régressions linéaires pou les trois conditions ($R > 70\%$ pour *Finger* et *Pointer*, $R > 85\%$ pour *Mouse*). Les zones ombrées représentent l'intervalle de confiance à 10% de ces régressions.

7.2.1.2 Évaluations qualitatives

L'expérience informelle avec *PhotoShuffler*, décrite plus haut (fig. 7.4 page 148), nous permet de faire les observations suivantes. Après un bref temps d'adaptation aux techniques d'interaction, les sujets utilisent l'interface avec facilité. Elles ignorent occasionnellement la présence de la caméra et se penchent au-dessus de l'interface. Les doigts sont alors cachés de la caméra et le suivi ne fonctionne plus. Le fait d'« oublier » le système perceptif est un résultat positif : il montre l'**autonomie** du service puisque les sujets ne ressentent pas le besoin conscient de coopérer avec le système. Le principal problème rencontré par les sujets est lié non pas à la perception mais à la technique d'interaction : pour « attacher » un doigt à une photo il est nécessaire de marquer une pause de 300 ms ce qui ne semble pas naturel et, même après apprentissage, reste contraignant. Dans 4% des cas les sujets effectuent un geste ballistique passant par la cible en augmentant apparemment la pression sur la surface au passage sur la photo. Le fait de reproduire des geste naturels sur la surface augmenté est également positif car il montre que le système perceptif favorise l'**immersion**.

Lors de l'expérience formelle effectuée avec *DoodleDraw*, 36 utilisateurs ont eu a mener à bien une tâche complexe en collaboration : la reproduction d'un graphe, contenant 17 segments de forme et de couleur différentes, en le dessinant avec leurs doigts sur une surface de dimensions A4. Le rappel moyen des sujets est de 88% (écart-type 6%)

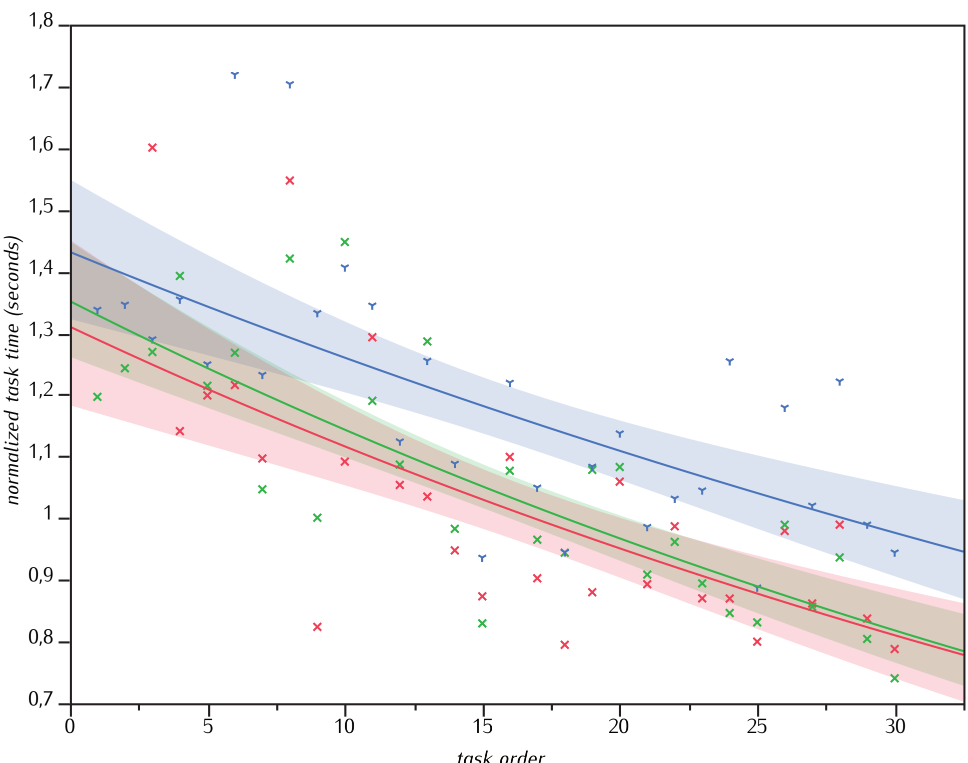


Figure 7.13 . Progression de la performance d'un utilisateur.

Ce graphique présente la progression de la performance de lensemble des utilisateurs pendant notre expérience. En abscisse, le temps discret (i.e. la successions des tâches élémentaires). L'intervalle réel est d'environ 5 s entre tâches successives. En ordonnées, le temps de réponse normalisé, défini comme le ratio du temps de réponse et de l'indice de difficulté. Le code couleur est repris à la figure 7.11 page 161.

Par souci de simplicité nous supposons que l'effet de l'apprentissage a se traduit par une décroissance logarithmique des temps de réponse (i.e. le logarithme des temps de réponse varie linéairement). Les courbes du graphique représentent les régressions logarithmiques correspondantes ($R > 70\%$ pour *Pointer*, $R > 80\%$ pour *Finger* et *Mouse*). Les zones ombrées représentent les intervalles de confiance à 10% de ces régressions.

[Borkowski, 2006]. Les sujets ont été limités par leurs capacités de collaboration et de mémoire, plus que par le système interactif.

Lors d'une manifestation publique (*Fête de la Science* d'octobre 2006), environ 120 visiteurs utilisent informellement *DoodleDraw*. Leur observation nous permet de résumer les principaux problèmes liés à des défauts du tracker :

– les sujets utilisent la main entière, plutôt que leurs doigts, pour interagir. Le service *FingerTracker+* permet le suivi de la main entière mais c'est à la charge du développeur d'application de mettre en place des techniques d'interaction adaptées.
– interactions avec l'ombre du doigt. L'utilisation d'une source infrarouge entraîne l'apparition d'ombres qui sont confondues par le système avec des doigts. Ceci peut être évité si la source de lumière est directionnelle et coaxiale avec la caméra. Cependant ce phénomène n'est pas perçu par les sujets comme un problème : ceux qui le découvrent l'exploitent occasionnellement pour interagir à distance de la surface, de façon à pouvoir observer l'image projetée sans être gênés par leurs main et bras.
– comportement erratique de la perception lorsque l'interface est quasi-totalement occultée. Des dispositifs de sécurité doivent être mis en place pour interrompre la perception lorsque trop de mouvement ou trop d'occlusions sont détectées.
– gestes trop rapides, en particulier le va-et-vient d'un doigt pour colorier une région. La caméra perçoit alors une image floue et le service perceptif ne peut pas fonctionner. Une solution est d'utiliser un éclairage plus puissant et de régler convenablement le temps d'exposition et l'ouverture de l'iris de la caméra.

7.2.1.3 Evaluations quantitatives

Nous évaluation la qualité de service du suivi de doigts possède selon quatre axes qui correspondent aux requis énoncés au chapitre 5 :

– nombre de doigts suivis : de 0 à 20 ;
– latence : inférieure à 50 ms ;
– précision et stabilité statique : 1 mm, ou 1 pixel de l'interface graphique ;
– autonomie : assisté à l'initialisation, coopérant en ligne.

Durant une expérience informelle nous avons demandé à deux groupes successifs de 6 utilisateurs d'utiliser *PhotoShuffler* pour organiser un ensemble de 24 photos selon les critères de leur choix. Nous avons observé jusqu'à 12 doigts interagissant simultanément. Dans le cas de *DoodleDraw*, l'IG exploite uniquement 2 doigts (en fait, un doigt par zone de l'interface : un pour la palette de couleurs, un autre pour le dessin).

La latence totale des système utilisé pour *PhotoShuffler* et *DoodleDraw* a été évaluée quantitativement, en observant le retard de la projection d'un retour visuel sur

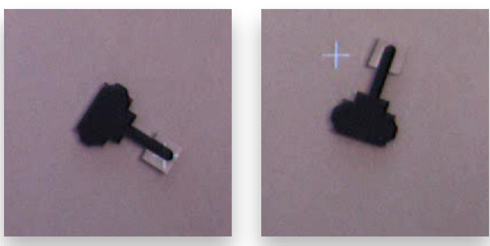


Figure 7.14 . Mesure de latence pour `gml.tracker.finger`

L'appareil, construit à l'aide de LEGO Mindstorms™, reproduit l'apparence d'un doigt. Il est donc reconnu est suivi par notre système. Il peut être mis en rotation. À gauche, l'appareil au repos. À droite, l'appareil en rotation. La croix blanche projetée est la sortie du service de suivi. Connaissant la vitesse de rotation, l'angle formé par la croix blanche, l'axe de rotation, et l'extrémité du pseudo-doigt, nous pouvons calculer la latence totale du système.
Cette technique de mesure est inspirée de [Liang et al., 1991].

l'agent d'interaction grâce à une caméra indépendante (figure 7.14 ci-contre). La valeur moyenne mesurée est de 80 ± 18 ms, pour le premier système et 67 ± 21 ms pour le second. La différence entre les deux systèmes est due à trois facteurs : le capteur utilisé (une Sony EVID-30P analogique associée à une carte d'acquisition Active Silicon LFG, puis une AVT Marlin Firewire de latence plus faible) ; la performance de la machine hôte des services (2× PowerPC 7400 à 1,45 GHz, puis 2× Pentium IV Xeon à 2,8 Ghz) ; et l'évolution de l'architecture (dans le second cas, l'acquisition video est traitée en parallèle avec les algorithmes de vision). La latence ne satisfait par le requis de 50 ms maximum imposée par le contrat (elle le dépasse respectivement de 60% et 34%). Nous avons cependant constaté plus haut que la dégradation d'utilisabilité n'est pas rédhibitoire : le système reste globalement utilisable. Nous avons observé que la latence augmente au plus de 15% lorsque le nombre de doigts suivi augmente.

La précision et la stabilité statiques sont déterminées par le seuil fixé lors de l'étape d'association du suivi (c.f. page 130). La détection étant précise à 0,75 pixel près dans la vue caméra, les événements ne sont émis que lorsque un mouvement détecté excède 1,5 pixels. Selon l'échelle et la définition de la vue ceci correspond à une précision de 3,5 mm (*PhotoShuffler*) et 1,2 mm (*DoodleDraw*). Ce résultat est supérieur respectivement de 350% et 20% à la précision imposée par le contrat.

Enfin, l'autonomie du système correspond au contrat : à l'initialisation le systèmes est assisté par l'utilisateur final pour régler la caméra. Le calibrage géométrique est automatique car il est assuré soit par `gml.calibrator.automatic`, soit par *PDSTracker*. En ligne le système coopère avec l'utilisateur final pour des opérations de remise à zéro ponctuelles (seulement pour *DoodleDraw*).

7.2.2 `gml.tracker.widgets`

Ce service n'a pas été le sujet d'expériences utilisateurs. Nous pensons cependant que, bien que ne remplaçant pas l'expérimentation, les enseignements de `tracker.finger` lui sont pour la plupart applicables. Nous avons mesuré quantitativement la qualité de service fournie par ce service.

Nous avons estimé la latence totale d'un système perceptif utilisant `tracker.finger`. Nous mesurons la part de la latence due à nos calculs (35 ms) et en déduisons la part due au matériel (32 ms). En utilisant le même matériel et en mesurant les temps de calul, nous pouvons alors estimer la latence totale d'un système utilisant `tracker.widgets`. Pour 60 striplets (10 boutons) la latence totale est de 52 ms. Le temps de pause pour activer un bouton étant réglé à 150 ms, ce service est donc 45% plus performant que `tracker.finger` pour les tâches de sélection. Il serait interessant de reproduire l'expérience utilisateur présentée ci-dessus pour ce service.

L'implémentation actuelle voit sa latence augmenter avec le nombre de striplets utilisés. Le service conserve une fréquence de traitement maximale (30 Hz) jusqu'à 300 striplets (50 boutons). Sa latence est alors de 65 ms. Un prototype d'implémentation utilisant un GPU d'entrée de gamme (nVidia Quadro NVS 280) permet de traiter jusqu'à 1300 striplets (environ 220 boutons) dans les mêmes conditions de latence et de fréquence.

La précision native du service est de 12 mm (les dimensions de la zone centrale d'un bouton SPOD. Nous avons assemblé un trackpad virtuel à l'aide de boutons SPOD afin d'estimer le maximum de précision que ce service peut atteindre. Des boutons SPODs son disposés sur une grille de 12×12 avec un pas de 4 mm, soit une surface tactile carrée de 48 mm de côté. Normalement ceci requiert 864 striplets (12×12×6). En exploitant les superpositions entre striplets, le temps de calcul nécessaire correspond à 108 striplets. La position en sortie du trackpad est le barycentre des striplets activés. Nous déterminons empiriquement une seuil de stabilité statique de 6 mm. Nous envisageons d'améliorer la précision de ce prototype jusqu'à environ 2 mm en calculant le barycentre pondéré par l'intensité de la réponse de chaque bouton.

L'autonomie du service est assurée par l'abscence de tout réglage. Lorsque ce service est utilisé la correction d'image automatique de la caméra peut être activée (gain, temps d'exposition, etc.) : le service est conçu pour être robuste aux variations d'éclairage. En pratique nous avons observé un fonctionnement correct du service (pas de fausses alarmes, peu d'échecs de détection) lors de changements naturels de l'éclairage (lumière extérieure).

7.2.3 Comparaison avec d'autres surfaces tactiles

Il nous paraît pertinent de citer ici quelques systèmes de perception (n'utilisant pas forcément la vision) qui sont susceptibles d'être utilisés pour remplir le même contrat que `tracker.finger` ou `tracker.widgets`. Certains sont des produits commerciaux, d'autres des prototypes de recherche. Nous résumons leurs caractéristiques sur le tableau ci-dessous. Certaines informations n'étant pas fournies par les auteurs, nous en donnons des estimations d'après notre connaissance des technologies employées. Les abréviations sont respectivement figure, nombre d'agents suivis, latence, précision, et autonomie :

Système	Fig.	#	Lat.	Préc.	Aut.	Remarques
FTIR Board	3.3	∞	70 ms	0,5 mm		projection par l'arrière
TouchLight	7.15	∞	70 ms	5 mm		projection par l'arrière
Canesta	7.16	1	10 ms	10 mm		surface limitée
SmartSkin	7.17	4	30 ms	2 mm		surface de taille fixe
DiamondTouch	7.18	8	20 ms	0,5 mm		2 doigts par utilisateur pas de confusion entre les doigts des utilisateurs
DViT	3.2	2	50 ms	1 mms		

En conclusion, les services que nous fournissons ont des performances comparables aux systèmes de perceptions actuels. Par rapport à ces systèmes, notre approche utilisant la vision par ordinateur présente les avantages suivants :

Figure 7.15 . La surface tactile *TouchLight*

D'après [Wilson, 2004]. À gauche, un schéma descriptif de l'installation. Derrière un écran semi-transparent un projeteur permet l'affichage. Une source de lumière infrarouge éclaire la scène. Deux caméras infrarouge observent l'arrière de l'écran. Au centre, les vues des deux caméras. Après projection dans un repère commun (celui de l'écran), les vues sont multipliées pour produire une carte de contact. Un algorithme de calcul de flot optique est apliqué au flux video résultant. À droite, une vue du système pendant l'interaction.

Figure 7.16 . Le clavier tactile de poche *Canesta*.

D'après [Roeber et al., 2003]. Un faisceau laser est orienté grâce à un miroir mobile. Il permet de projeter une interface graphique. À la base de l'appareil une diode infrarouge émet dans le plan de la table. Une caméra infrarouge détecte l'intersection de doigts avec ce faisceau.

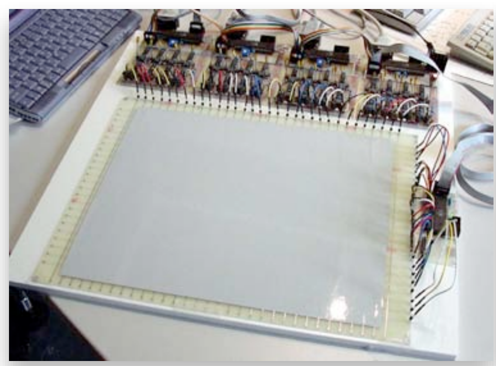
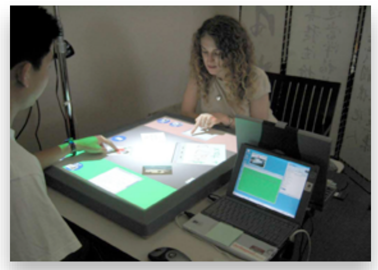


Figure 7.17 . La surface interactive *SmartSkin*

D'après [Rekimoto, 2002]. *SmartSkin* est une grille d'antennes qui détecte des variations de capacité. Elle fournit une carte bi-dimensionnelle qui associe à chaque position sur la table une mesure de distance des objets au-dessus de la table, jusqu'à une distance de quelques centimètres. En fixant un seuil sur cette mesure, elle permet de détecter et localiser le contact avec la peau.
La précision de la grille est faible (1 centimètre), mais par interpolation les auteurs parviennent à une précision de l'order du millimètre pour le suivi. La latence du système perceptif est d'environ 30 ms soit deux fois moins que notre solution utilisant la vision.



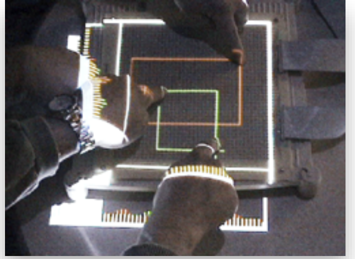


Figure 7.18 . La table *DiamondTouch*

D'après [Dietz et Leigh, 2001] et [Diaz-Marino et al., 2003]. À gauche, deux utilisateurs utilisant l'application *UbiTable* sur la table *DiamondTouch*. À droite, deux sujets interagissent indépendemment sur un prototype de la table.

- il n'est pas nécessaire d'équiper la surface interactive (une table ou un tableau peuvent être utilisés) ; en particulier, la surface peut conserver son rôle premier (par exemple, il est toujours possible de dessiner sur un tableua blanc).
- la taille de la surface n'est contrainte que par la définition de la caméra et la puissance de calcul de l'ordinateur qui exécute le service perceptif.
- le nombre de points de contact n'est pas limité.
- l'encombrement est faible, puisque le matériel est déporté.

7.3 Évaluation de l'architecture

Notre approche structurelle a été validée par le déploiement de nombreux prototypes utilisant *gmlVision* en particulier dans des environnements hétérogènes (i.e. en collaboration avec d'autres « services » n'utilisant pas la même architecture).

Manque de métriques d'évaluation.

Comme nous l'avons expliqué en introduction de ce chapitre, les méthodes d'évaluation éxpérimentales ou numériques ne sont pas applicables ici (notamment faute de moyens). En outre, dans le cadre d'une revue et typologie des méthodes d'évaluation existantes [Babar et al., 2004] montre que ces méthodes fournissent des définition de la notion d'architecture logicielle incompatibles entre elles. Il paraît dès lors difficile d'appliquer l'une de ces méthodes.

Une évaluation expérimentale « de terrain » ne nous est pas non plus possible. Rappelons les trois critères isolés par [Klemmer et al., 2004] pour ce type d'évalution : facilité d'utilisation, facilité de réutilisation, support de patrons de conception. Pour évaluer la boîte à outils vis-à-vis de ces critères il serait nécessaire faire interprêter les rôles de Stanislas, Patrick, et Laurence à un ensemble de sujets. Ceci nous est impossible faute de temps.

Évaluation analytique.

Faute de métriques utilisables, nous choisissons d'adopter une démarche d'évaluation analytique. Dans la littérature, nous pouvons isoler des critères permettant de qualifier une « bonne » architecture de services de vision pour l'interaction. La piste la plus interessante nous semble être de revenir aux critères de Myers, présentés en 4.1.2 page 65.

Les exemples de code présentés au chapitre précédent (page 122) indiquent que le middleware *gmlBIP* et le framework de construction de services ont un seuil fonctionnel bas : le nombre de ligne de code nécessaire pour assembler un service ou exploiter un service est très faible ($\approx$ 10 lignes en Tcl). D'autre part, ils offrent un chemin de moindre résistance pour des tâches usuelles (construction de services, connection à un service) et des besoins spécifiques déjà identifiés (adaptation de services, aggrégation de services).

Le critère du *viser juste* (décrit en 4.1.2.1 page 65) est d'après nous le plus important pour évaluer l'architecture de notre boîte à outils. Il faut non seulement satisfaire le besoin (ce que nous avons fait en identifiant et respectant les requis d'asynchronisme, abstraction, isolation et contrat) mais aussi ne pas satisfaire *plus* que le besoin. Nous rappelons au lecteur que les services de notre boîte à outils ne sont que des outils pour Laurence, parmi d'autres : ils ne doivent pas empiéter sur ses autres outils. Pour évaluer le *viser juste*, nous nous tournons vers la littérature du domaine de l'architecture logiciel.

Adéquation structurelle de *gmlVision*.

Les auteurs de [Garlan et al., 1995] décrivent un défaut des architectures logicielles concrètes qu'ils nomment *architectural mismatch* (inédéquation structurelle). D'après nous, il s'agit de ce que Myers nommerait « mal viser » (le contraire de « viser juste »).

Ce concepte englobe les problèmes qui surviennent lorsqu'un concepteur utilise des « pièces existantes » (telles que *gmlVision*) pour construire un système.

Parmi les symptômes de l'*architectural mismatch*, on peut citer : un volume de code excessif, de mauvaises performances, la nécessité de modifier les « pièces » existantes, ou la nécessité de réimplémenter certaines fonctions des « pièces ». Tout développeur utlisant des bibliothèque tierce a eu l'occasion de les expérimenter.

D'après les auteurs, il existe quatre suppositions qui sont les causes de ce problème. Nous tentons de démontrer ici que de telles suppositions, bien que faites lors de la conception et de l'implémentation de notre boîte à outils, ne constituent pas un obstacle à l'interopération.

– nature des composants. Nous supposons que tous les composants d'un système sont des services, c'est-à-dire qu'ils répondent aux quatres contraintes : asynchronisme, isolation, abstraction, et contrat. Elles nous paraissent correspondre à de bonnes de bonnes pratiques de développement logiciel ; cependant, il est trivial de concevoir de systèmes n'y répondant pas à partir des composant les satisfaisant. Par exemple, des appels synchrones peuvent être construits à partir de couples requètes-réponse asynchrones : il suffit que le client « attende » de manière bloquante la réponse ; dans `tracker.finger`, le couple `ResetQuery - Reset` peut être considéré comme synchrone. Enfin, un composant qui n'est pas un service peut tout de même interopérer avec les services de *gmlVision* (comme cela a été le cas avec *Digitable*).
– nature des connecteurs. Nous supposons que toutes les communications entre composants ont lieu sur des canaux `BIP/1.0`. En utilisant *gmlBIP* ou une autre implémentation, il est aisé de construire des adaptateurs entre un canal `BIP` et un autre type de connecteur. Par exemple, *jOMiSCID*, une implémentation de `BIP`, est capable de faire le pont entre notre architecture et le système à composants répartis OSGi.
– structure de l'architecture globale. Nous supposons que l'ensemble du système est une fédération *peer-to-peer* de services `BIP`. Comme le montre l'exemple paquetages de services fournis par *gmlVision*, il est cependant facile de construire un monolithe à partir de services encapsulés dans un seul programme.
– processus d'assemblage : la nature *peer-to-peer* et sans état des services fait qu'ils sont indifférent à l'ordre dans lequel ils sont instanciés et connectés.

Pour conclure, et bien qu'une évaluation de terrain de la boîte à outils soit indispensable, nous pensons avoir fourni des pistes qui indiquent que *gmlVision* satisfait les critères de Myers, c'est-à-dire (à notre sens) que son architecture est utile et utilisable.

7.4 Conclusion

Dans ce chapitre, nous avons validé et évalué les propositions du chapitre 5 et la réalisation présentée dans le chapitre 6.

Au travers du déploiement de divers systèmes interactifs sur la base de *gmlVision* nous avons amélioré de façon itérative notre approche de conception et démontré que l'architecture orientée-services est adaptée à la construction de tels systèmes.

L'étude de ces mises en œuvre, du point de l'utilisateur, met en évident des défauts de la boîte à outils. Pour la rendre plus utilisable il sera nécessaire de fournir des services « support » manquants, d'identifier des patrons de construction et d'assemblage d'applications, et de fournir des chemins de moindre résistance correspondants sous forme de services supplémentaires, ou d'API adaptées dans le middleware. Enfin, il sera nécessaire de rendre le comportement dynamique des applications (flux d'événements, état des connexions) mieux observable.

Les services de *gmlVision* permettant d'implémenter des surfaces interactives tactiles ont été évalués quantitativement et qualitativement : ils remplissent en partie les contrats énoncés au chapitre 5, et satisfont aux besoins correspondants identifiés

au chapitre 3. Nous appliquons pour les évaluer une démarche typique en génie logiciel, inspirée du modèle en V (présenté en 5.1.2.4 page 91) : effectuer une analyse quantitative de chaque service, un test utilisateur avec une application minimale, et des expériences utilisateur sur des applications réelles. Finalement, la comparaison avec les surfaces tactiles existantes est favorable. Les autres services n'ont pas été implémentés ou pas été évalués.

En termes d'interface de la boîte à outils, le prototype DoodleDraw permet une conclusion positive : le seuil fonctionnel est bas (le concepteur de l'application de dessin a fourni peu d'efforts pour accéder aux services), et le plafond est élevé : l'application est différente de ce qui a été observé dans l'état de l'art ; elle met également en oeuvre des services tiers (i.e. qui ne font pas partie de *gmlVision*).

Enfin, nous avons évalué informellement et de manière analytique, dans la mesure du possible, notre proposition d'architecture concrète fondée sur `BIP/1.0` et son implémentation *gmlBIP*, en utilisant les critères de Myers pour l'utilisabilité d'une boîte à outils.

8 Conclusion : contributions, limitations et perspectives

« The realistically biggest challenge to contextual and perceptual interfaces [is to bridge] the barriers between the disciplines working on these technologies—specifically, human-computer interaction, speech recognition, and computer vision. While there are small communities working on the boundaries (...) folks working on recognition seldom pay attention to context or the applications that come later. We'll make some progress that way, but if we want a revolution, which the market is ready for, then we need to forget tribal allegiances and work together. »

[Canny, 2006]

Afin de promouvoir la démocratisation des interfaces de nouvelle génération fondées sur la perception visuelle, nous proposons un ensemble de concepts, méthodes, et outils permettant de concevoir et réaliser des services de vision par ordinateur réellement utilisables en interaction Homme-machine.

Après avoir définit notre problème, nous en présentons une analyse fonctionnelle et une analyse structurelle. Nous dégageons une ensemble de requis de ces analyses, puis nous présentons une solution méthodologique et technique. Finalement, nous évaluons quantitativement et qualitivement notre solution.

Nous présentons ici un résumé de nos contributions, puis un ensemble de perspectives ouvertes par nos travaux.

8.1 Contributions

Nos contributions s'orientent selon trois axes : contributions conceptuelles, méthodologiques, et techniques.

Contributions conceptuelles.

Nous réalisons un état de l'art des systèmes interactifs de « nouvelle génération », c'est-à-dire s'écartant du paradigme WIMP, à partir duquel nous identifions un ensemble de besoins de perception visuelle artificielle. Nous proposons une taxonomie qui permet de décrire ces besoins. Elle définit quatre axes : la description de l'agent interactif, la ou les propriétés de l'agent d'intérêt pour l'interaction, la qualité de service requise, et l'environnement d'usage. Cette taxonomie permet aux fournisseurs de logiciels perceptifs (spécialistes en vision par ordinateur) et à leurs utilisateurs (spécialistes en interaction Homme-machine) d'exprimer, dans une *lingua franca*, les services fournis par les uns et requis par les autres.

Afin de permettre de concevoir et réaliser une boîte à outils utile et utilisable, nous identifions quatre classes d'utilisateurs : l'utilisateur final (à qui nous ne nous adressons pas directement), le développeur de nouvelle interactions (notre cible prioritaire), le développeur de services de perception, et le développeur d'algorithmes de vision par ordinateur.

Une analyse centrée sur les besoins des ces quatre classes d'utilisateur nous permet d'identifier d'une part les requis fonctionnels de notre bibliothèque (exprimés dans notre taxonomie), d'autre part un ensemble de requis structurels quantitatifs (latence, autonomie, fiabilité) et qualitatifs (asynchronisme, isolation, abstraction, et contrat).

Contributions méthodologiques.

Du point de vue du réalisateur de systèmes interactifs, nous proposons de concevoir des services de vision *ad hoc* destinés à une tâche d'interaction particulière plutôt que des briques logicielles élémentaires et génériques. Nous montrons pourquoi l'approche générique n'est pas adaptée au domaine de la vision par ordinateur pour l'interaction Homme-machine. La définition de l'ensemble des services *ad hoc* qui doivent être fournis dans notre boîte à outils est un problème difficile. Nous proposons une démarche itérative et empirique pour définir cet ensemble afin de couvrir les besoins fonctionnels identifiés lors de notre état de l'art et représenté par notre taxonomie. Cette démarche est fondée sur l'itération de cycles de développement, déploiement, et évaluation des services.

Concernant la validation et l'évaluation d'un service perceptif, nous proposons de vérifier que la fourniture du service est en accord avec son *contrat* exprimé dans notre taxonomie. Nous proposons d'évaluer les services perceptifs par des expérimentation utilisateur, ce qui est une pratique originale dans le domaine de la vision par ordinateur.

Contributions techniques.

Notre bibliothèque de services extensible *gmlVision* fournit (a) des composants de vision par ordinateur et (b) plusieurs services prêts à l'emploi pour la perception et le support de la perception et interaction Homme-machine. Les services finalisés permettent la création de surfaces aumgentées tactiles, ils sont : `gml.tracker.finger` qui permet la détection et le suivi précis de doigts nus, mais ne fonctionnant qu'en environnement contraint ; `gml.tracker.widgets`, qui implémente les widgets tactiles, activés par l'occlusion par un doigt, robuste aux variations d'éclairage et utilisable sur une surface mobile ; `gml.grabber.camera`, une abstraction des pilotes d'acquisition qui permet une acquisition video performante en utilisant une API normalisée ; enfin, `gml.calibrator.automatic` se charge du calibrage géométrique précis d'une surface augmentée.

La conception de ces services a été l'occasion de contributions mineures au domaine de la vision par ordinateur. Il s'agit d'une part des techniques pragmatiques utilisées dans nos services pour détecter les doigts et rendre une surface tactile : le seuillage heuristique sur l'histogramme, les filtres de forme à rejet rapide (FRF), et les *striplets*, des détecteurs d'occlusion élémentaires. Une autre contribution est la technique employée pour calibrer géométriquement une surface quelconque : le calibrage automatique utilisant la détection d'une mire asymétrique par différence d'image.

Sur le plan structurel nous proposons le protocole d'interopération *peer-to-peer* orienté services `BIP/1.0`. Il est conçu en visant un seuil d'apprentissage bas mais également un plafond fonctionnel élevé. Les performances visées doivent satisfaire les requis de l'interaction Homme-machine (en particulier la latence). Nous proposons une implémentation de `BIP/1.0` : le middleware *gmlBIP* implémente une architecture concrète de type SOA 2.0. Il permet d'implémenter des services (avec ou sans *gmlVision*) et d'interopérer avec des services `BIP`.

8.2 Limitations et perspectives

Notre boîte à outil a déjà été utilisée par quelques développeurs qui ne sont pas liés à son développement (jouant chacun des rôles d'intérêt : Laurence, Patrick, et Stanislas) ; l'évaluation de son utilité et de son utilisabilité n'est donc que partielle. Notre travail s'inscrit dans un processus de développement centré utilisateur : à l'instar de tout système interactif, la validation ultime est liée à un large déploiement et usage du système. Notre priorité est donc maintenant de distribuer notre boîte à outils. Son utilisation par d'autres chercheurs nous conforterait dans notre approche et nous permettrait d'en apprendre davantage sur ses adéquations et inadéquations aux besoins. La distribution de *gmlVision* passe par un effort important de préparation : préparations de distributions logicielles pour les différentes plate-formes, documentation des services, réalisation d'un site web pour diffuser l'information. Cet effort d'ingénierie est à la frontière d'un travail de *recherche* mais il nous semble être le seul moyen d'obtenir un point de vue objectif et fiable sur la qualité de notre travail. C'est pourquoi nous avons déjà largement avancé cette préparation : la boîte à outils est déployée sous plusieurs systèmes d'exploitation (MacOS dans l'équipe IIHM et Linux dans l'équipe PRIMA) et un site web documente son utilisation (`iihm.imag.fr/letessier/gml`).

Après ce point de vue général, nous passons en revue les limites et perspectives de notre travail d'un point de vue technique, méthodologique et conceptuel.

Limitations et perspectives techniques.

Remarquons tout d'abord que les limites de qualité de service sont éphémères. D'après nous, l'utilisation de caméras à faible latence (par exemple les Guppy de la compagnie Allied Vision Technologies), combinée au calcul sur carte graphique et à un videoprojecteur rapide permettront prochaienement de franchir la limite des 50 ms de latence. L'exploitation de caméras ayant des définitions plus élevée permettra d'augmenter la précision des services de suivi.

Concernant les services déjà présents dans *gmlVision*, les limitations les plus contraignantes sont liées à un environnement d'exécution réduit. Ces limites sont dues essentiellement à la difficulté des problèmes de vision par ordinateur sous-jacents. Cependant, la vision est un domaine qui évolue très vite et qui bénéficie directement des progrès en terme de capacité de calcul des ordinateurs. Nous voyons ici de nombreuses voies de recherche et de progression.

Nous pensons en particulier que le service de suivi de doigts peut être considérablement amélioré. Les possibilités de calcul massivement parallèlle sur carte graphique rendent possible la mise en oeuvre d'algorithmes plus complexes. Nous envisageons de remplacer notre technique de segmentation simpliste par la projection sur des *eigenbackgrounds*, un modèle statistique de fond prometteur [Oliver et al., 2000]. D'après ses auteurs, ce modèle permet d'absorber le bruit du capteur, de gérer plusieurs conditions de lumière discrète ainsi que la mobilité de l'interface, et il serait capable d'éliminer les ombres.

Le service fournissant des « widgets virtuels » couvre un environnement d'usage déjà large mais qui pourrait être étendu davantage. Nous avons l'intention d'explorer d'autres fonctions-réponse pour les *striplets* que nous espérons plus discriminantes. Nous souhaitons également rechercher des méthodes pour optimiser automatiquement les calculs en exploitant les recouvrements de *striplets*. Nous espérons pouvoir obtenir un service d'une précision comparable à celle du suivi de doigts actuel.

Hormis les services existants, nous avons proposé des API pour plusieurs services qui n'ont pas été réalisés dans *gmlVision*. Les plus importants, c'est à dire ceux qui permettront de réaliser la plus grade variété de systèmes, sont les suivants :

– `gml.tracker.tag`. Il existe plusieurs solutions logicielles pour la détection et le suivi de *tags*. Le plus répandu, l'ARToolkit, souffre d'une architecture monolithique. Il est en « inadéquation structurelle » avec la possibilité de le réutiliser dans une autre architecture. Une autre approche est fournie par ARTag [Fiala, 2005]. Il

serait intéressant d'explorer la possibilité d'empaqueter ARTag dans un service de *gmlVision* ou s'il est nécessaire de redévelopper ce service.

– `gml.identifier.object`. Le problème de l'identification d'objets ne possède pas encore de solution reconnue en vision. Cependant, certaines approches récentes de reconnaissance fondées sur les points d'intérêt SIFT [Lowe, 1999] et le modèle d'apparence 2dPCA [Yang et al., 2004] paraîssent prometteuses. Une évaluation de ces approches est nécessaire afin de déterminer si le service offert correspond aux besoins que nous avons identifiés.

Nous identifions également deux autres fonctions importantes qui font défaut à *gmlVision* : le calibrage automatique des paramètres d'acquisition video et une mesure de l'environnement d'usage afin de contrôler si le service est dans les limites opérationnelles de son contrat. Notre pensons que ces deux fonctions sont des voies de recherches qui auront un impact positif sur la robustesse et sur l'agrandissement du champ d'application des systèmes perceptifs visuels. En ce qui concerne le calibrage automatique, nous souhaitons explorer la piste de l'optimisation par descente de gradient de l'entropie de l'histogramme de luminosité (tel qu'évoqué en 7.1.2.1 page 157).

Limitations et perspectives méthodologiques.

Les résultats obtenus dans les premières utilisations de *gmlVision* vont dans le sens de la validation de notre approche (choix d'une interface spécifique plutôt que générique ; démarche mixte, empirique et itérative fondée sur le modèle en V). Cependant, nous pensons poursuivre l'application de notre démarche dans l'objectif de réaliser le recouvrement fonctionnel des besoins et de rendre les services déjà présents plus recouvrants (c'est-à-dire améliorer leur qualité de service tout en les rendant robuste à un environnement plus large). Nous pensons que la difficulté pour atteindre ces objectifs sera un indice du bien-fondé de notre démarche.

Par ailleurs, nous manquons de méthodes et de patrons de conception d'applications. De tels patrons sont indispensables pour fournir un chemin de moindre résistance à Laurence. Si grâce à `BIP` l'assemblage d'une fédération de services est facilité, il reste *ad hoc* : il est effectué de manière potentiellement différente pour chaque application. Là encore, la distribution et l'utilisation de *gmlVision* sont indispensables pour identifier les pratiques des systèmes de vision pour l'interaction, en observant des utilisateurs ayant le rôle de Laurence.

La structure de notre boîte à outils, et des fédérations de services permettant de construire une application, est également un sujet que nous souhaitons approfondir. Les machines modernes étant de plus en plus parallèles (les processeurs typiques de haut de gamme possèdent 4 coeurs. D'autres architecture standard, comme CELL, en possèdent 9). L'exploitation de ce parallèlisme devient une nécessité. Notre architecture à services en bénéficie naturellement : chaque service étant indépendant, ils peuvent être exécutes simultanément. Nous pourrions cependant explorer les possibilités offertes par ce parallèlisme à un niveau d'abstraction plus bas, celui des composants constituants de services.

Limitations et perspectives conceptuelles.

Nous avons vérifié que notre taxonomie couvre l'expression des besoins des systèmes interactifs étudiés dans l'état de l'art. Il est certain que cet état de l'art, de même que les besoins de perception, sont amenés à évoluer avec les avancées dans le domaine des interactions innovantes. Nous ne considérons pas notre taxonomie comme un ensemble figé, mais plutôt comme un outil destiné à être complété pour décrire les nouveaux besoins et concerver ainsi son utilité en tant qu'outil de communication.

Nous avons fait le choix d'une taxonomie peu formelle. Ce choix est en adéquation avec le rôle principal de la taxonomie d'outil de communication entre clients et fournisseurs de services de perception. Mais ce manque de formalisme présente le risque d'une interprétation ambiguë. Nous envisageons donc de corriger cette taxonomie au fur et à mesure des problèmes rencontrés. En effet, l'approche alternative de rendre la taxonomie plus formelle se ferait au détriment de se simplicité d'utilisation.

La partie de la taxonomie concernant la qualité de service pourrait faire l'objet d'une notation plus formelle, et d'une syntaxe interprétable par la machine. Par exemple, l'utilisation de *gmlVision* est envisagée dans le contexte des environnements intelligents : dans une pièce augmentée, de nombreuses caméras peuvent être présentes ; la disponibilité d'un contrat électronique peut alors permettre la sélection automatique du « bon » capteur pour un service interactif donné.

8.3 Conclusion

Le besoin de boîtes à outil de service de vision pour l'interaction est flagrant mais il ne semble pas satisfait. Les volontés extérieures d'utiliser les services de *gmlVision* se sont faites sentir dès les premières présentations du système de suivi de doigts. Nous considérons donc la distribution de *gmlVision* comme une étape importante de nos travaux. Pourtant, nous n'avons pas encore franchi ce pas, et ce malgré l'insistance de nos contacts en laboratoire de recherche comme dans d'autres structures.

Ceci s'explique par notre volonté de fournir un outil utile et utilisable pour la communauté de l'interaction Homme-machine. L'état de l'art montre que les quelques tentatives dans ce domaine n'ont pas été des succès et ont eu tendance à associer une mauvaise réputation à la vision par ordinateur en interaction Homme-machine. L'objectif essentiel de notre travail doctoral a donc été d'identifier les *bonnes pratiques* qui permettront, à terme, une distribution d'une boîte à outils adaptée au besoin. L'effort d'analyse des besoins, principes, méthodes, techniques et outils a donc primé sur l'effort d'implémentation ou de distribution.

Au-dela de la distribution de *gmlVision*, nous espérons que nos travaux constituent une fondation interessante : pour reprendre les termes et l'analogie historique de Turk, si la vision pour l'interaction est actuellement dans son « Âge de pierre », nous espérons que des approches et des outils similaires aux notres puissent faire entrer ce domaine de recherche directement dans l'Époque moderne.

A Le protocole BIP : spécification et implémentation

Nous présentons la conception et les motivations à l'origine de BIP dans la section 6.1.1 page 118. Il a, à l'origine, été introduit dans l'article [Borkowski et Letessier, 2006]. D'autres auteurs décrivent des implémentations différentes : [Emonet et al., 2006] et [Reignier et al., 2006].

La suite de cette annexe reproduit la spécification du protocole, telle qu'utilisée pour implémenter *gmlBIP*.

BIP/1.0: Basic Interconnection Protocol for Event Flow Services

Author: Julien Letessier
Author: Dominique Vaufreydaz
Author: Sebastien Pesnel
Author: Remi Emonet
Revision: 47
Status: Draft
Date: September 6, 2006

Abstract

This document describes low-level and high-level aspects of the `BIP/1.0` protocol that allows the cooperation of services used to build interactive systems.

Contents

1 Introduction

2 Terminology

3 Services overview

- 3.1 Discovering a service
- 3.2 Connecting to a service
- 3.3 Service parameters

4 Communicating with services

- 4.1 Low-level message format
- 4.2 Establishing a link
- 4.3 Communicating over TCP
- 4.4 Communicating over UDP
- 4.5 Communicating over multicast UDP

5 Inspection and control of services

1 Introduction

BIP is designed to meet a number of requirements encountered when constructing interactive systems:

function reuse and distribution Subsystems may need to be used by multiple other subsystems, possibly running on different machines (for perfomance or geographic reasons).

For instance, a video capture service might be used concurrently by a surveillance system located on a centralized server, a video-conference application and a motion capture service, both co-located to the camera.

low threshold, high ceiling Easy integration of BIP services into heterogeneous systems is a requirement. Minimalistic implementations (under 100 lines of code in any language) should always possible, in order to allow for easy interoperability.

performance Interactive systems always place a constraint on overall latency.

For example, in the case of a finger tracker service used for interaction on an augmented surface (an example of tightly coupled interaction), the latency must be under 50 ms to maximize usability. On the other hand, for a service that counts the number of persons present in a room, a latency of 1 to 5 seconds might be acceptable.

robustness In some cases, information needs to be reliably conveyed between subsystems, i.e. any message sent by a service must arrive to a listening peer (possibly within a tolerable time limit).

This may be the case, e.g. for the person counter described above.

These requirements have a number of consequences that influence how BIP is designed:

- it is presumed that the systems communicating using BIP are isolated, black-box services;
- no assumption is made on the peer operating systems, languages, or method of communication (machine-local or distant);
- services must use user-friendly naming, because interconnection will be performed by humans in the general case;
- peer services must be dynamicaly discoverable in order to alleviate the need for address-based and port-based setup;
- multiple transports must be available to cope with the multiple latency and reliability needs;
- the protocol must contain a minimal portion that is sufficient to make services useful, while allowing simple programs like `telnet` to interoperate with any BIP implementation.

The minimal BIP protocol only consists in an exchange of ASCII messages with 1-line message headers over a TCP socket. This allows for minimal interoperability with a very low effort. Implementors of BIP can enrich the inteconnection by adding support for UDP communications, multicast UDP, DNS-based dynamic discovery, and finally service control through an XML control protocol.

2 Terminology

Service An opaque processing unit that accepts given input events and produces given output events. It possesses a number of state variables (variable values) and properties (constant values), altogether refered to as *parameters*. Interaction with a service is acheived by establishing a *link* with one or several of its *channels*.

Peer An entity on the network that can communicate with a service: e.g. another service, an architecture adapter, or a user application.

Channel A named point of communication between a service and one or several peers. One or more links can be established with a channel. There are three type of channels, depending on the direction of the flow of data: input channels, output channels, and duplex channels. Note that this is different from system-level sockets, e.g. TCP or UDP sockets.

Link A means of sending messages to a service or receiving messages from a service. A link is comparable to an established socket-based network connection, except that no assumption is made on the underlying transport (TCP, UDP, multicast UDP).

Message An encapsulation of one or several events for transmission over a link.

Event A discrete piece of information that transits between a service and a connected peer.

3 Services overview

3.1 Discovering a service

`BIP` services advertize themselves using the DNS-SD [1] convention over Multicast DNS [2]. They use the `local.` domain, TCP transport, and the `bip` service name; in other words, fully qualified names for `BIP` services names must be as follows:

```
<name>._bip._tcp.local.
```

where `<name>` is chosen by the service instanciator. The TCP port number advertized in the DNS-SD SRV record is the port used for control channel to the service (see below).
The DNS-SD TXT record fields listed below must or may be present for each service. They must not change during the lifetime of a service.

id (**required**)

The peer id used to identify this service during communcations with other peers.

class (**optional**)

The class of service. This will be standardized in a future revision of `BIP`. Classes starting with a period are reserved.

owner (**optional**)

A descriptor of who owns the service. This can be used to limit connections to a cluster of services, e.g. in the case of concurrently running systems that uses the same services. It is advised to use either the login or the full name of the user who instantiated the service.

Additionally, there must exist a field for each channel. The field key is the channel name, and the field value is of the form `port/type`, where `port` is the channel's TCP port, as a decimal number, and `type` is one of `i`, `o` or `d`, for input channels, output channels, and duplex channels, respectively.
Example record:

```
id=FADA97CE
class=bip.source.noise
owner=mezis
events=123/d
noise=456/o
```

Notes:

- TXT record field labels (e.g. "inputs") are case-insensitive. It is recommended to only use lower-case letters, numbers, hyphens and underscores in labels.
- Channel names (e.g. "events", "noise") should be short human-readable names. They must be formed only of lower-case letters, digits, hyphens and underscores.

3.2 Connecting to a service

There are four possible types of channels within a service:

- the *control channel* vehicles events that allow a peer to inspect a service and control its state variables;
- *input channels* transmit events to one of a service's inputs;
- *output channels* transmit events from one of a service's outputs.
- *duplex channels* are used for transactions between peers, for instance as a support for remote method invocation support.

A service accepts TCP link on one TCP port for control links, and one TCP port for each input and output. For inputs, it also accepts messages over UDP if a TCP link has been established (see below). For outputs, it can emit over UDP if requested.
Data that transits over any link with service obeys the message format described below (in Low-level message format). For the control link, data inside messages obeys further specification (see the sections on inspection and control below).

3.3 Service parameters

Services may define any number of parameters.
All services must define at least the following so-called "universal" parameters.

status (**integer**)

Possible values are

0. the service is not running;
1. the service was asked to run, but is waiting for one or more inputs to be connected;
2. the service is running.

[1] DNS-Based Service Discovery: browsing and discovery of services using DNS queries. Described at http://www.dns-sd.org/.

[2] Multicast DNS: performing peer-to-peer DNS queries over IP Multicast. Described at http://www.multicastdns.org/.

lock (**integer**)

Possible values are

- zero; or
- the ID of a peer connected to the control port.

If non-zero, the service will not modify any parameter upon reception of a control query that originates from a peer which ID is not equal to the value of this parameter. A control answer event will still be emitted.

This parameter is automatically reset to zero if the corresponding peer disconnects.

4 Communicating with services

Motivation: the communication protocol must not impose any software requirements on the peers that connect to a service.

4.1 Low-level message format

Messages are the lowest-level of the protocol, i.e. what transits over a network connection to a service. They contain a fixed-size (34 bytes) header and a variable-size payload; the header is the concatenation of the following ASCII strings:

1. Magic header (7 bytes). It is a string of the form `BIP/X.Y`, where `X` is the major version of the protocol, `Y` is the minor version. All peers should emit `BIP/1.0`, and accept any form `BIP/1.Y`.
2. Space (1 byte).
3. Peer identifier (8 bytes). A unique identifier for the peer that emitted the message. It represents a 32-bit unsigned integer formatted as hex.

   For peers that are services, the first 16 bits may represent the service startup time in seconds since the UNIX epoch, modulo 65536, and the last 16 bits may be randomly chosen [3].
4. Space (1 byte).
5. Message identifier (8 bytes). A message identifier that is unique for the (simplex) link. It represents a 32-bit unsigned integer formatted as hex. It is incremented for each sent message and starts from 0. The message identifier 0 is used during the link establishment process (see Establishing a link).
6. Space (1 byte).
7. Payload size (8 bytes). The number of bytes in the message contents. It represents a 32-bit integer formatted as hex.

[3] In order to prevent identifier collisions, a service should use the best random algorithm available. In particular, on UNIX platforms, the random number generators in the standard C library should be seeded, e.g. with the service startup time.

Both the header and the payload are immediately followed by the <cr> carriage-return plus <lf> line-feed line termination sequence (bytes `0x0A` and `0x0D`), which are not taken into account for length calculations. In the case of an empty payload (length zero), the termination sequence must still be appended; in other words, the message end with two successive <cr><lf> sequences.

Example message:

```
BIP/1.0 DEADBEEF 00000000 0000000<cr><lf><cr><lf>
BIP/1.0 DEADBEEF 00000000 000000D<cr><lf>hello, world!<cr><lf>
```

Note that the format of message contents is not specified for input, or output duplex links to a service. However, it is specified for the control connections (see the sections on inspection and control below).

It is strongly advised to keep the format of message contents for a particular service as simple as possible, for the sake of inter-operability with other systems and services. In particular, service designers should:

- avoid using high-level languages except where relevant: XML is pertinent for large tree-like structures, not for transmiting two integers;
- transmit plain ASCII text instead of binary when possible: it will be easier to trace the protocol for debugging purposes;
- use <cr><lf> as a line delimiter even inside messages.

4.2 Establishing a link

A service must accept only one *control* link at a time. A service may choose to only accept one link to any *input*, *output* or *inoutput* channel at a time.

To establish a link to a service, a peer must:

1. Open a TCP connection to one of the service's ports (either the control port or one of the input, inoutput, or output ports);
2. Send an initial message to the service, with message number zero;
3. Receive an initial message from the service, with message number zero.

These two messages allow the peers to identify eachother, thanks to the peer ID provided with each message. The order of steps 2 and 3 is not specified. These initial messages may be empty or contain one or several lines of meta-information about the link, the channel, or the service.

For instance, suppose peer A with peer ID `DEADBEEF` opens a TCP connection to peer B with peer ID `DADABABA`. Peer A then sends the message:

```
BIP/1.0 DEADBEEF 00000000 00000000<cr><lf><cr><lf>
```

to peer B, and peer B sends the message:

```
BIP/1.0 DADABABA 00000000 00000000<cr><lf><cr><lf>
```

to peer A. The link is then considered to be established.

Link meta-information is formatted a la RFC822, in successive key-value pairs. Keys must be formed of lower-case letters, digits and hyphens. Values must be formed of printable 7-bit ASCII characters. Key and value are separated by a colon and any number of space or tab characters. Successive key-value pairs are separated by the <cr><lf> line delimiter. Example (delimiters ommited):

```
BIP/1.0 DEADBEEF 00000000 00000066
service-name:   Dummy Service
channel-name:   Dummy output channel
udp-port:       21587
status:         Up and running
content-type:   text/xml
```

A number of keys are reserved (though optional):

- `content-type` is the MIME format of all messages. If not specified, the default if `text/plain`.
- `udp-port` can be used to establish the UDP part of a link (see Communicating over UDP);
- `multicast-group` can be used to establish the multicast UDP part of a link;
- if present, `service-name` must be the service name;
- if present, `channel-name` must be the channel name.

4.3 Communicating over TCP

Communicating over TCP is possible for any service link. Once a link has been established as described above, communication is acheived by sending and receiving any number of successive messages.

It is recommended that services that require low-latency communication over TCP send and receive messages (e.g. empty messages) at all times. This is to ensure that the TCP window stays as open as possible, and that occasional "bursts" of data are not limited by the enpoints' TCP stacks.

The actual minimal data throughput to maintain depends on you local implementation(s) of TCP; it is however recommended that you maintain an overall average throughput of half your maximal throughput between the two peers.

4.4 Communicating over UDP

Communicating over UDP is possible for any link (for service input, output and inoutput channels), except control links. A "source" peer may send messages over UDP to a "destination" peer if

a. a link has been established (over TCP) as described above; and

b. the destination UDP port is known. This is possible if the "destination" peer included the `udp-port` key in its initial message.

Each UDP datagram must contain exactly one message.

A service should ignore incoming UDP messages from a peer if no link is established.

4.5 Communicating over multicast UDP

The above paragraph (Communicating over UDP) remains valid for communication over multicast UDP.

No protocol is specified when choosing a multicast group. It is recommended to select a group at random.

5 Inspection and control of services

Service inspection is achieved by sending `controlQuery` events and receiving `controlAnswer` events over a control link to a service. The TCP port for the control link to a service can be determined using the service discovery mechanism. Inspection provides information on *service parameters*, *inputs* and *outputs*. Control allows to modify the values of a service's parameters.

Services not implementing the control protocol must emit the following message immediately after any connection has been established with their control port:

```
<controlError id="00000000" type="not-implemented"/>
```

Communication over the control channel obey the companion XML Schema in `bip-control.xsd`.

For instance, the following query and answer show how to inspect a service parameter:

```
<controlQuery id="DADADEAD">
        <variable name="stars"/>
</controlQuery>

<controlAnswer id="DADADEAD">
        <variable name="stars">
                <value>123</value>
                <default>100</default>
                <type>integer</type>
                <access>read</access>
                <description>current number of
                   stars in the sky</description>
        </variable>
</controlAnswer>
```

The control protocol is still subject to change and is not normative.

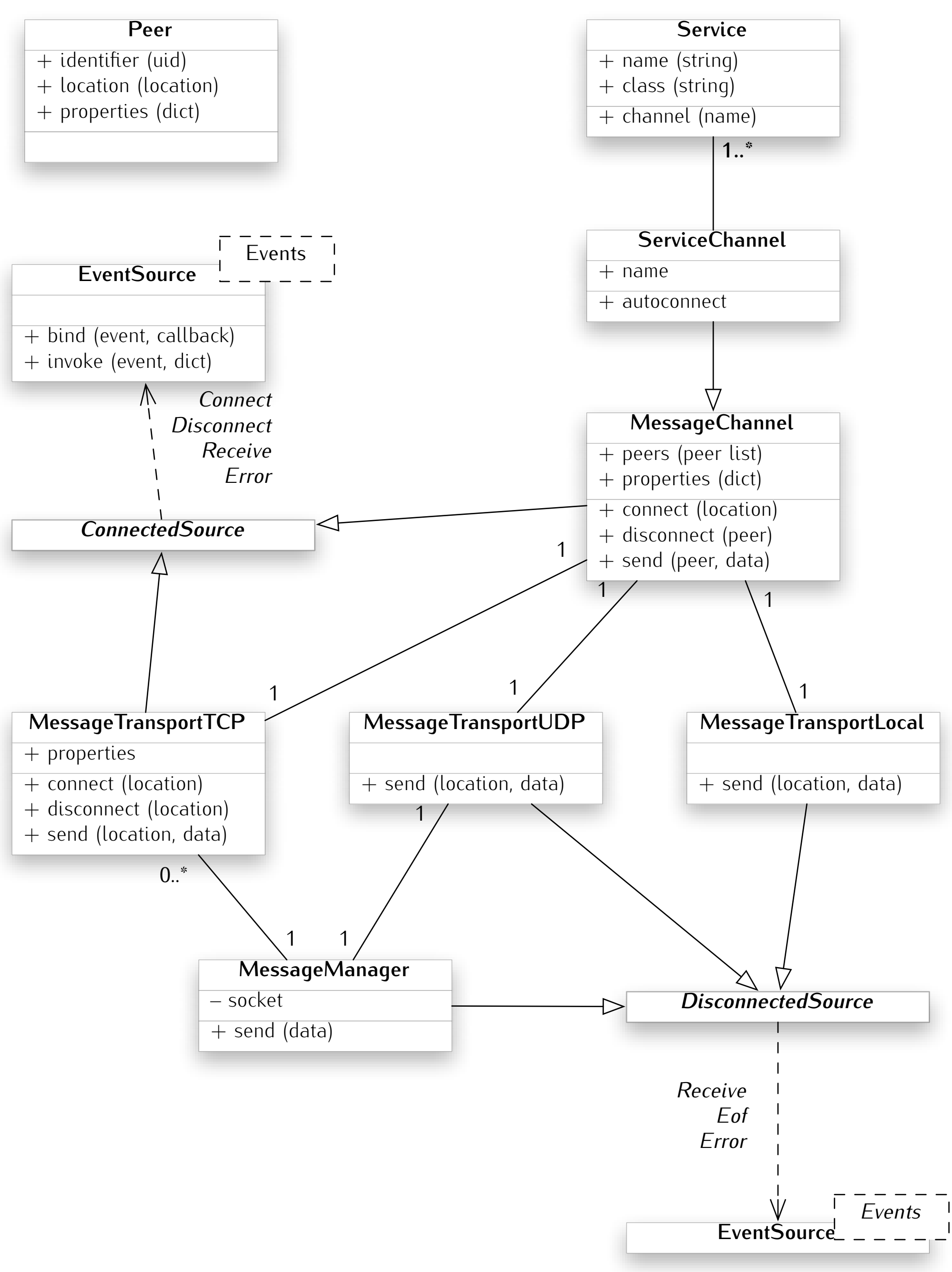


Figure A.1 . Diagramme de classes UML de l'implémentation de référence de `BIP/1.0`, *gmlBIP*.

Légende des types de données utilisés :

uid	nombre entier unique
string	chaîne de caractères
event	chaîne de caractères (un nom d'événement)
dict	tableau associatif (métadonnées d'un événement)
data	bloc de données non typé
callback	procédure ou méthode d'un objet
location	un couple hôte, numéro de port où l'hôte est soit un FQDN, soit une adresse IP

B Outils avancés pour la vision par ordinateur

à compléter

Dans cette seconde annexe, nous présentons deux aspect technologiques de *gmlVision*, liés aux contraintes de la vision par ordinateur temps réel : la métaprogrammation des primitives de traitement d'image d'une part, et l'implémentation d'algorithmes de vision sur carte graphique d'autre part.

B.1 Métaprogrammation statique des primitives de traitement d'image

Écrire des programmes de traitement d'image est répétitif, et source d'erreurs. Pour reprendre les exemples du chapitre 6, un programme de convolution par un noyau gaussien et un autre effectuant une érosion morphologique sont très proches : seules 5% de lignes de code sont fonctionnellement différentes. En outre, un même algorithme devra souvent être ré-implémenté pour chaque espace de couleur utilisé. Enfin, de nombreuses erreurs proviennent de manipulations mémoire qui peuvent être difficiles à déceler.

Étant donné que nous devons écrire de nombreuses primitives de traitement (*gmlVision* en contient 55), nous décidons donc d'explorer des techniques nous permettant de rendre leur développement plus fiable et plus rapide, sans compromettre la performance (critique) du résultat.

Les outils que nous proposons répondent aux contraintes suivantes :

- offrir une interface externe commune pour exploiter les algorithmes ;
- offrir une interface interne commune pour l'implémentation ;
- réduire l'effort de développement ;
- ne pas déteriorer la performance.

B.1.1 Techniques et outils existants

[Köthe, 2000]
[RWTH, 2005]
[EPITA, 2005]

B.1.2 Utilisation de l'outil Lg dans *gmlVision*

B.1.2.1 Description de Lg

un outil de *templating* (application de patron).

permet de définir un squelette de code (en l'occurence pour le traitement d'une ou plusieurs images), et de ne laisser à l'utilisateur que la définition de la partie pertinente du code : le traitement d'un pixel particulier.

B.1.2.2 Génération de code pour l'imagerie

unification de l'API

mise en commun de fonctionnalités

optimisations :

- code spécifique pour chaque encodage
- code spécifique pour chque configuration
- déroulage de boucles automatique
- prefetching automatique

B.2 vision par ordinateur sur cartes graphiques

revue des approches, possibilités [Owens et al., 2007]

approches haut niveau : utiliser un langage traditionnel, avec quelques extensions. la carte graphique n'est alors qu'un *backend* d'exécution

[du Toit et Marier, 2006]

[Buck et al., 2004]

approches bas niveau : extensions d'OpenGL. accès aux parties programmables standardisées du *pipeline* de traitement graphique des GPUs : *vertex shaders* et *fragment shaders*

[NVIDIA, 2006]

[Kessenich et al., 2006]

Un avantage de Cg / GLSL : on peut rester dans le langage de haut niveau (script) pour expérimenter sur le traitement d'image

- réduction du cycle de développement
- moins de code binaire
- possiblités de métaprogrammation dynamique

C Formalisation des besoins rencontrés

Dans cette annexe, nous proposons une représentation des besoins de service rencontrés lors de l'état de l'art du chapitre 3, en utilisant le formalisme de notre taxonomie. Le lecteur remarquera que certains cartouches peuvent être sujets à débat : en effet, lorsque les auteurs des articles décrivant les systèmes correspondants fournissent insuffisamment de détails, nous avons complété l'expression du besoin au mieux de nos connaissances.

S1 — détection et identification d'un objet plan parmi des objets connus

agent		*qualité de service*		*image & cadrage*		*eclairage*	
objet plan		#agents	0 à 2	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	100%	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☐☒☐			contexte	☐	variation	☐

S2 — détermination des positions relatives d'objets plans

agent		*qualité de service*		*image & cadrage*		*eclairage*	
objet plan		#agents	0 à 2	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	100 mm	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☐☒☐			contexte	☐	variation	☐

S3 — suivi d'un objet plan en deux dimensions

agent		*qualité de service*		*image & cadrage*		*eclairage*	
objet plan		#agents	0 à 1	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm	couleur	visible	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☐☒☐			contexte	☐	variation	☐

S4 — détection et suivi d'objets colorés de forme simple

agent		*qualité de service*		*image & cadrage*		*eclairage*	
post-it		#agents	0 à 10	définition		structure	
propriété d'intérêt		latence		résolution	2 mm	variation	
aspect		précision	1 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☒☐☐☐			contexte	☐	variation	☒

S4' — détection et suivi d'objets colorés de forme simple

agent		*qualité de service*		*image & cadrage*		*eclairage*	
jeton rouge		#agents	0 à 10	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☒☐☐☐			contexte	☐	variation	☒

S5 — numérisation de l'aspect d'un objet plan

agent		*qualité de service*		*image & cadrage*		*eclairage*	
post-it		#agents	1	définition		structure	
propriété d'intérêt		latence		résolution	250 μm	variation	
aspect		précision	250 μm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☐☐☐			contexte	☐	variation	☐

S6 — identification visuelle d'un objet plan

agent		*qualité de service*		*image & cadrage*		*eclairage*	
post-it		#agents	1	définition		structure	
propriété d'intérêt		latence		résolution	250 μm	variation	
aspect		précision	n/a	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☐☐☒			contexte	☐	variation	☐

S7 — suivi d'un doigt par utilisateur

agent		*qualité de service*		*image & cadrage*		*eclairage*	
doigt		#agents	0 à 4	définition		structure	
propriété d'intérêt		latence		résolution	2 mm	variation	
aspect		précision	1 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☒☐☐☐			contexte	☐	variation	☐

S8 — identification d'un doigt

agent		*qualité de service*		*image & cadrage*		*eclairage*	
doigt		#agents	0 à 20	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	n/a	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☒☐☐			contexte	☐	variation	☐

S9 — suivi avec orientation des doigts

agent		*qualité de service*		*image & cadrage*		*eclairage*	
doigt		#agents	0 à 5	définition		structure	
propriété d'intérêt		latence		résolution	500 μm	variation	
aspect		précision	1mm/5°	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☐☐☐			contexte	☒	variation	☐

S10 — suivi des doigts de plusieurs utilisateurs

agent		*qualité de service*		*image & cadrage*		*eclairage*	
doigt		#agents	0 à 20	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☒☐☐☐			contexte	☐	variation	☐

S11 — suivi/posture des mains

agent		*qualité de service*		*image & cadrage*		*eclairage*	
main		#agents	0 à 4	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	10 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☐
identité	☐☒☐☐			contexte	☐	variation	☐

S12 — suivi/classification d'objets

agent		*qualité de service*		*image & cadrage*		*eclairage*	
objet		#agents	0 à 100	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	10 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☐☒☐☐			contexte	☒	variation	☐

S13 — suivi/classification d'objets

agent		*qualité de service*		*image & cadrage*		*eclairage*	
objet		#agents	0 à 100	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	10 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☐	complexité	☒
identité	☐☒☐☐			contexte	☒	variation	☐

S14 — détection d'occlusions

agent		*qualité de service*		*image & cadrage*		*eclairage*	
doigt		#agents	0 à 1	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	10 mm	couleur	infrarouge	*confusion*	
géométrie		autonomie		mobilité	☒	complexité	☒
identité	☐☐☐☐			contexte	☒	variation	☒

S15 — suivi 3D du visage

agent		*qualité de service*		*image & cadrage*		*eclairage*	
visage		#agents	1	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	10 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité	☒	complexité	☐
identité	☐☐☐☐			contexte	☒	variation	☒

S16 — suivi 3D d'une surface rectangulaire

agent		*qualité de service*		*image & cadrage*		*eclairage*	
surface		#agents	0 à 1	définition		structure	
propriété d'intérêt		latence		résolution	1 mm	variation	
aspect		précision	1 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité		complexité	
identité				contexte		variation	

S17 — détection de la silouhette

agent		*qualité de service*		*image & cadrage*		*eclairage*	
corps		#agents	0 à 1	définition		structure	
propriété d'intérêt		latence		résolution	10 mm	variation	
aspect		précision	10 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité		complexité	
identité				contexte		variation	

S17' — détection de la silouhette

agent		*qualité de service*		*image & cadrage*		*eclairage*	
corps		#agents	0 à 1	définition		structure	
propriété d'intérêt		latence		résolution	10 mm	variation	
aspect		précision	1 mm	couleur	couleur	*confusion*	
géométrie		autonomie		mobilité		complexité	
identité				contexte		variation	

D Glossaire des acronymes

Nota bene. Certaines des définitions succintes données ici sont copiées ou traduites de l'encyclopédie numérique *Wikipedia* (http ://wikipedia.org).

ACC *Analyse en composantes connexes*. Technique d'analyse d'image qui détermine les ensembles de pixels « d'un seul tenant » dans une image noir et blanc, ainsi que leurs propriétés (centre, aire, etc.)

API *Application Programmer Interface*. L'interface utilisateur d'une boîte à outils ligocielle.

APR *Apache Portable Runtime*. Une bibliothèque multi plates-forme fournissant des services de bas niveau, en particulier des IPC.

AR *Augmented Reality*, Réalité augmentée. Décrit les systèmes où des objets ou des images du monde réel voient leur aspect, leur fonction ou leur rôle augmenté par de l'information numérique.

ASCII *American Standard Code for Information Interchange*. L'encodage numérique des caractères de l'alphabet latin le plus répandu.

BIP *Basic Interconnexion Protocol*. Un protocole de communication *peer to peer* (égal à égal) conçu et implémenté dans le cadre de cette thèse.

BLAS *Basic Linear Algebra Subroutines*. Un ensemble de routines qui effectuent les opérations élémentaires de l'algèbre linéaire, tel que les multiplications entre vecteurs et matrices, et dont l'interface est standardisée.

CAO *Conception Assistée par Ordinateur*. Domaine des logiciels et des techniques permettant de concevoir, de tester, et de réaliser des outils et des produits manufacturables.

CLIPS Communication Langagière et Interaction Personne-Système. Un laboratoire de recherche en informatique situé à Grenoble.

COM *Component Object Model*. Une plate-formet logicielle pour le développement à composants introduite par Microsoft™ en 1993.

CORBA *Common Object Request Broker Architecture*. Une architecture logicielle pour le développement de composants répartis. C'est un standard maintenu par l'association *Object Management group*.

CSCW *Computer Supported Cooperative Work*. Travail collaboratif assisté par ordinateur, un domaine de recherche proche de l'IHM.

CV *Computer Vision*, vision par ordinateur.

DCE/RPC *Distributed Computing Environment / Remote Procedure Calls*. Une architecture concrète à composants distribués fortement couplée, comparable à CORBA. Implémentée et utilisée dans Windows (MSRPC).

DCOM *Distributed Component Object Model*. Une extension de COM pour des composants répartis sur plusieurs machines. Un compétiteur de CORBA.

DNA *Distributed Network Architecture*. Une boîte à outils logicielle standardisée pour la conceptioon d'application répartie, qui permet en particulier l'implémentation d'une architecture à composants répartis.

dpi *dot per inch*, pixel par pouce. Unité de mesure de résolution ; typiquement 92 pour les écrans et 600 pour les imprimantes laser.

DV *Digital Video*. Un format vidéo répandu qui permet d'enregistrer des vidéos sur des cassettes en numérique avec une faible compression pour chaque image. C'est le standard de fait de l'édition numérique.

EDA *Event Driven Architecture*, architecture dirigée par les événements. Décrit une structure asynchrone, où les éléments (composants) sont faiblement couplées et communiquent par échanges d'événements discrets, indépendants de leur état.

EDL *Edit Decision List*. Une manière de représenter le montage d'une bande video. Elle contient une liste ordonnée de positions et de *timecodes* représentant où chaque clip video peut être obtenu pour réaliser le montage.

ESB *Enterprise Service Bus*. Une famille de middleware permettant à des applications hétérogènes d'interagir au travers de services standards qu'elles mettent à disposition. Fortement lié aux web-services et à la plate-forme Java.

FAME *Facilitator Agent for Multimodal Environments*. Projet européen de recherche, dont les thèmes incluent l'interaction Homme-machine. Utilise, en particulier, la vision (*Table Magique*).

FPS *First Person Shooter*. Jeu de tir à la première personne. Variété de jeu favorisant l'immersion visuelle de l'utilisateur.

FTIR *Frustrated Total Internal Reflection*. Phénomène optique qui provoque la diffusion du flux lumineux parcourant un guide d'ondes, aux points de contact avec un matériau d'indice de réfraction plus élevé que le milieu ambiant.

GLUT *Graphics Library Utility Toolkit*. Le standard de fait pour bâtir des applications interactives utilisant OpenGL pour le rendu graphique.

GPS *Global Positioning System*. Système de positionnement global, permettant de déterminer (en extérieur) la position absolue d'un recepteur sur terre (latitude, longitude et altitude).

GUI *Graphical User Interface*. Interface utilisateur graphique.

HCI *Human Computer Interaction*. Interaction homme-machine.

HID *Human Interface Device*. Périphérique interactif (d'entrée, de sortie, ou les deux).

HTTP *Hyper Text Transfer Protocol*. Le protocole de communication sous-tendant le Web.

IDL *Interface Description Language*. Langage utilisé pour décrire formellement une API, théoriquement indépendemment d'un langage de programmation

IHM Selon le context, interaction Homme-machine ou interface homme-machine.

IP *Internet Protocol*. Protocole de communication gérant le transport et le routage des paquets de données sur des réseaux informatique interconnectés.

IPC *Inter-Process Communication*. Ensemble des méthodes permettant à deux processus logiciels de communiquer. IPC de bas niveau : segments de mémoire partagée, sémaphores, verrous, queues de messages, ou sockets TCP/IP. À haut niveau : RPC, CORBA.

ISO *International Organization for Standardization*

JAI *Java Advance Imaging*

JPEG *Joint Photographers Experts Group*. Commision ayant conçu et implémenté le format de représentation d'images homonyme.

LAPACK *Linear Algebra PACKage*, une bibliothèque logicielle pour le calcul numérique, écrite à l'origine en Fortran77, et dont l'API est très largement utilisée.

OASIS *Organization for the Advancement of Structured Information Standards*, un consortium qui guide le développement, l'adoption et la standardisation de technologies centrées sur l'*e-business* et les services web.

O3MISCID *Open object-oriented middleware for service connection, inspection and discovery*. Une implémentation de `BIP/1.0`, fournissant des fonctionnalités supplémentaires pour l'intégration d'applications.

OpenGL *Open Graphics Library*. Une spécification standard définissant une API universelle pour produire des graphismes 2D et 3D.

ORB *Object Request Broker*. Un élément de middleware qui permet à un objet d'appeler une méthode d'un autre objet, éventuellement depuis un programme ou une machine différente.

PAL *Phase Alternating Line*, un encodage analogique d'un flux video couleur. Désigne également la définition de l'image véhiculée, 576 lignes par 720 colonnes.

PC *Personnal Computer*. Micro-ordinateur dont le prix, la taille, et les fonctionnalités sont pertinents pour un usage individuel et/ou par le grand public.

PDA *Portable Digital Assistant*. Ordinateur de poche, de taille et de performances réduites par rapport à un PC.

PDS *Portable Display Surface*, surface portable d'affichage. Interface mobile conçue par Stan Borkowski [Borkowski et al., 2003].

PTK *Peripheral Display Toolkit*. boîte à outils logicielle pour la conception d'application utilisant l'interaction ambiante. [Matthews et al., 2004]

QoS *Quality of Service*, qualité de service

QVGA *Quarter Video Graphics Array*. Désigne une image dont la définition est 320×240 pixels, souvent utilisée pour les écrans de PDA ou de téléphones mobiles, ou encore comme définition d'acquisition pour les webcams.

RFID *Radio-Frequency Identification*, une méthode pour stocker et récupérer des données à distance en utilisant des marqueurs plats, de petite taille, appelés « radio-étiquettes ».

RMI *Remote Method Invocation*. Technique permettant d'exécuter le code d'une méthode à distance, c'est-à-dire depuis un autre programme ou une autre machine. Terme utilisé presque exclusivement pour la plate-forme Java.

RPC *Remote Procedure Call*. Technique permettant d'éxécuter une procédure dans un autre espace d'adressage, sans que le programmeur n'explicite les détails de l'interaction à distance.

SDK *Software Development (Tool)Kit*. Ensemble d'outils de développement qui permettent de créer des applications pour un logiciel ou un matériel donné.

SGBDR Système de gestion de bases de données relationnelles. Ensemble logiciel permettant la manipulation de bases de données, structurées suivant les principes de l'algèbre relationnelle. Les plus connus sont Oracle, MySQL, ou Microsoft SQL Server.

SOA *Service Oriented Architecture*, architecture orientée service.

SOAP *Simple Object Access Protocol*, un protocole pour l'échange de messages XML entre machines, normalement véhiculé par HTTP. Standardisé par la W3C. Utilisé comme couche de base pour la construction de services web

SPOD *Simple Pattern Occlusion Detector*. C.f. 6.2.2 page 132.

SQL *Simplified Query Language*. Langage standardisé permettant l'interaction avec un SGBDR.

SVM *Support Vector Machine*, une technique de discrimination. Elle consiste à séparer deux (ou plus) ensembles de points par un hyperplan.

TIFF *Tagged Image File Format*. Format de fichier permettant de stocker des images. Permet de stocker différentes précisions (de 1 à 48 bits par canal), avec ou sans compression.

TCP *Transmission Control Protocol*, un protocole de transport fiable, en mode connecté, utilisé généralement sur IP. Il permet l'échange d'un flux d'octets entre applications.

UDP *User Datagram Protocol*, un protocole de transport non fiable, en mode non connecté, utilisé généralement sur IP. Il permet la transmission discrète de paquets taille limitée. Sa latence est plus faible que celle de TCP.

UI *User Interface* ou *User Interaction*.

UNIX un système d'exploitation créé en 1969 ; par abus de langage, la famille de systèmes en dérivant, comme GNU/Linux ou MacOS X.

USB *Universal Serial Bus*, un bus informatique série plug-and-play servant

à brancher des périphériques à un ordinateur.

VEIL *Vision Event Interpratation Layer*. C.f. figure 6.13 page 132.

WIMP *Windows, Icons, Menus, Pointer*. Paradigme classique de l'interaction Homme-machine. C.f. 3.1 page 36.

XML *eXtensible Markup Language*, un langage de balisage générique, standardisé par le W3C. Les langages dérivés sont spécifiés par un *XML Schema* (en XML) ou une *Document Type Declaration* (en SGML).

ZNCC *Zero Normalized Cross Correlation*. Une technique de mise en correspondance ou reconaissance d'images, partiellement robuste aux variations d'éclairage.

E Dépendances logicielles de *gmlVision*

La bibliothèque *gmlVision* est volumineuse : elle contient environ 100 000 lignes de code source, dont 10% sont écrits en Lg, 20% en Tcl/Tk, et 70% en C et C++.

Nous avons cependant cherché à réduire cet effort de développement en réutilisant des éléments logiciels existants. Ceci nous permet de bénéficier de l'expertise d'autres developpeurs dans des domaines où ne nous sommes pas experts, et de préserver la portabilité de notre boîte à outils.

Nous présentons dans cette annexe les différents outils que nous avons exploité, en expliquant leur intérêt.

Tcl et Tk.

Comme expliqué au cours de ce document, l'emploi d'un langage interprêté de haut niveau offre de nombreux avantages pour l'intégration d'applications et la construction d'interface graphiques, et accélère l'ensemble du processus de développement. Cependant, les langages interprêtés (comme Java, Python, ou Tcl) sont bien moins performants que ceux de bas niveau (C/C++, Fortran, ou Ada). Pour des opérations intensives en calcul et/ou en volume de données, comme le traitement d'images, la différence varie d'un à deux ordres de grandeur (respectivement pour Python et Tcl). Nous avons donc choisi d'implémenter les opérations coûteuses en C, avec une API la plus simple possible, et de les rendre exploitables dans un langage de haut niveau.

Tcl semble un choix adapté, pour plusieurs raisons. L'extension du langage par ajout de fonctions écrites en C est simple ; l'emballage d'une procédure demande typiquement l'écriture d'une dizaine de lignes de code. Ce processus peut en outre être automatisé à l'aide d'outils qui interprètent le code de bas niveau et produisent un *wrapper* (code d'emballage) pour le langage de haut niveau. Un tel outil est SWIG (*Simple Wrapper Interface Generator*).

Tcl est développé et distribué avec *Tk*, une boîte à outils de construction d'interfaces graphiques. Avec Java, c'est à notre connaissance le seul langage qui présente cette caractéristique.

Tcl et *Tk* sont portables, et fonctionnent sur l'ensemble des plates-formes logicielles utilisées à l'heure actuelle ; en particulier Windows, GNU/Linux, et MacOS.

Enfin, l'interface et l'implémentation de *Tcl/Tk* est mûre : son utilisation est répandue depuis une quinzaine d'années, ce qui en fait un environnement de développement stable et efficace. En termes d'utilisabilité : *Tcl/Tk* possède une grande amplitude plancher-plafond fonctionnel.

Abstraction du système d'exploitation : APR, Bonjour.

Si *Tcl* fournit de nombreuses abstractions du système d'exploitation, elles ne sont pas toutes suffisantes pour notre usage. Les mécanismes d'IPC fournis par *Tcl*, en particulier, ne permettent pas d'implémenter `BIP/1.0`. Nous utilisons d'autres bibliothèques pour satisfaire les besoins de communication ; nos critères de sélection sont, par ordre de priorité, l'utilité et l'utilisabilité de la bibliothèque ; se portabilité ; et sa maturité.

Après une revue de l'existant, nous choisissont d'utiliser deux outils : APR et Bonjour. APR (*Apache Portable Runtime*) est la bibliothèque chargée de l'abstraction dans Apache, le principal serveur HTTP utilisé. Elle fournit, en particulier, des services d'IPC (sockets TCP et UDP, verrous locaux et globaux, et gestion de segments de mémoire partager) et de chargement dynamique de code. Bonjour implémente DNS-SD sur UDP-multicast, et permet la publication et la découverte de services (sur le réseau local).

Exploitation du GPU : OpenGL, GLEW.

Dans notre cadre applicatif, utiliser la puissance de calcul de la carte graphique moderne (GPU) possède deux avantages : (*a*) elle permet d'afficher de nombreux flux video simultanés, par exemple dans un moniteur de service, et (*b*) elle peut effectuer certains traitements d'image bien plus rapidement qu'un processeur généraliste (CPU).

La bibliothèque standard qui permet d'accéder aux fonctionnalités d'un GPU est OpenGL. Microsoft fournit une bibliothèque concurrente, DirectX, mais elle ne fonctionne que sur la plate-forme Windows.

Il existe plusieurs versions d'OpenGL (les plus répandues sont 1.1, 1.2, et 2.0) ; les pilotes de GPU n'implémentent qu'une version. Hors, selon les versions d'OpenGL et l'implémentation, l'API des fonctionnalités que nous exploitons sont soit absentes, soit présentes directement dans OpenGL (par exemple les *fragment programs* dans OpenGL 2.0), soit dans une extension (par exemple `ARB_fragment_program`). Afin de s'abstraire de la version d'OpenGL utilisée par le client, et des extensions présentes ou non, nous utilisons la bibliothèque GLEW (*OpenGL Extension Wrangler*), qui fournit une API unifiée.

Calcul numérique : GSL, BLAS.

Certaines des opérations indispensables dans *gmlVision* demandent du calcul matriciel relativement complexe : le calcul de la matrice de calibrage, les transformations de coordonnées.

Les BLAS fournissent des procédures qui permettent de réaliser ces calcul, et possèdent des implémentations performantes sur toutes les plate-formes (par `Accelerate.framework` sur MacOS, ATLAS sous Linux, et IPP sous Windows). Ces procédures sont cependant de bas niveau : nous utilisons GSL (*Gnu Scientific Library*), une bibliothèque à l'API simple qui abstrait les BLAS et fournis des services de plus haut niveau (décompositions matricielles, calcul de valeurs et vecteurs propres, etc.)

Chargement et sauvegarde d'images : FreeImage.

Pour effectuer des tests unitaires, ou tester un service perceptif sur un flux video enregistré, nous devons fournir la possibilité de lire et d'enregistrer des images dans les formats standard : au moins TIFF, JPEG, et PNG. Chacun de ces formats dispose d'une implémentation de référence (`libtiff` pour TIFF, etc.) ; leurs API étant disparates, nous ne voulons pas fournir l'effort de développement considérable pour obtenir une API commune. La bibliothèque libre multi plate-formes FreeImage remplit cet office.

Acquisition video.

Abstraire la capture du flux video est un problème épineux, car les API utilisées sont très différentes ; il existe en outre une API « standard » par plate-forme, plus des API spécifiques à une norme, voire à un produit. Nous utilisons les API QuickTime (sous MacOS) et Video For Linux 2 (sous Linux) pour permettre l'accès à la plupart des modèles de caméras et cartes d'acquisition. Nous implémentons également notre API abstraite au-dessus de `libdc1394`, une bibliothèque performante pour l'accès aux caméras sur Firewire, qui fonctionne sur les deux plate-formes.

Bibliographie

[Accot et Zhai, 2002]
Johnny Accot, Shumin Zhai. *More than dotting the i's — foundations for crossing-based interfaces*. In CHI '02 : Proceedings of the SIGCHI conference on Human factors in computing systems, pages 73–80. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-453-3.
doi : http ://doi.acm.org/10.1145/503376.503390. Cité page 161.

[Allan et al., 1995]
Vicki H. Allan, Reese B. Jones, Randall M. Lee, Stephen J. Allan. *Software pipelining*. ACM Computing Surveys, volume 27(3) :pages 367–432, 1995. ISSN 0360-0300.
doi : http ://doi.acm.org/10.1145/212094.212131. Cité page 156.

[Angerson et al., 1990]
E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, D. Sorensen. *LAPACK : A portable linear algebra library for high-performance computers*. In Proceedings of Supercomputing '90, pages 2–11. 1990. Cité page 89.

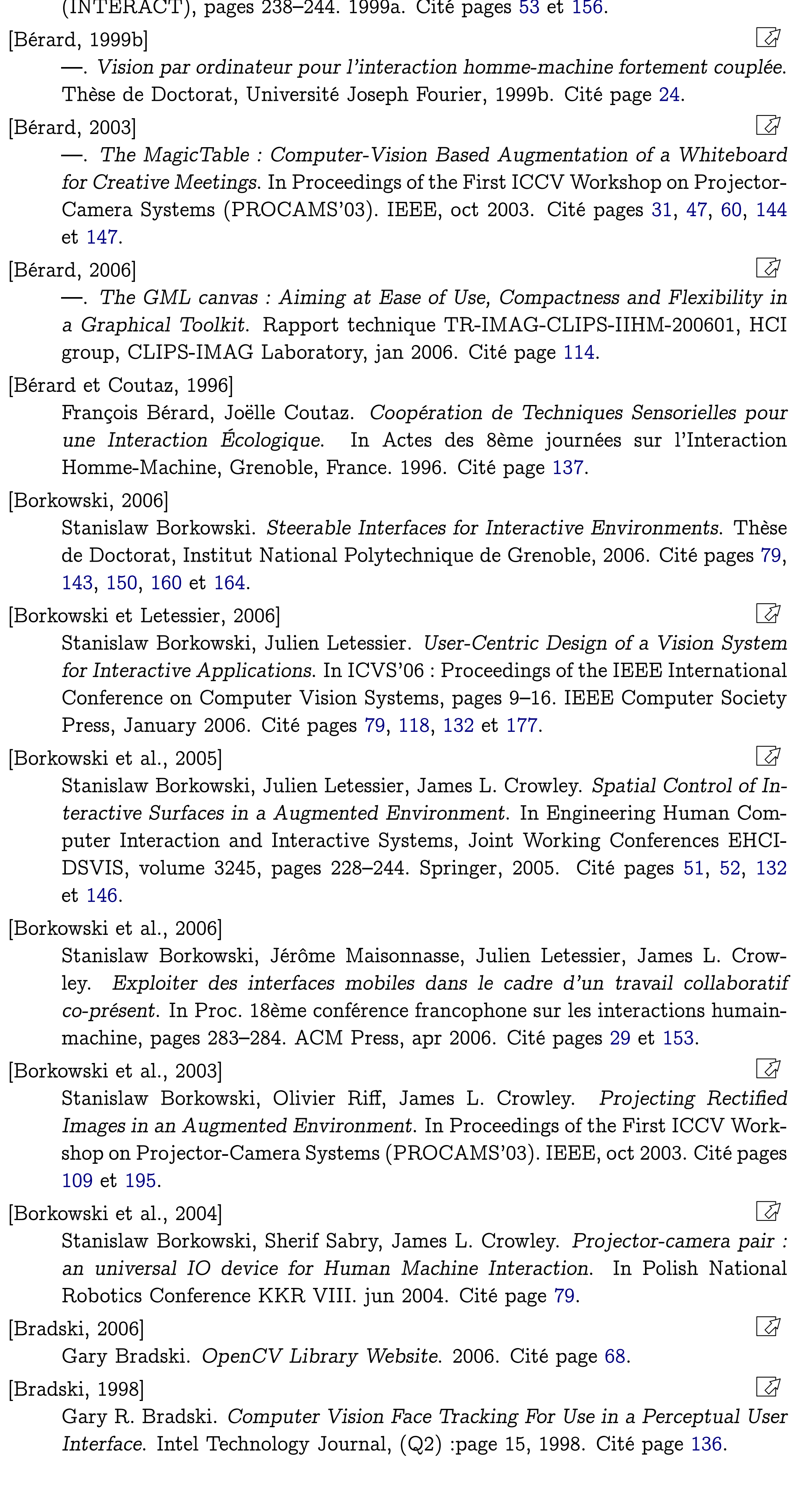
[Appert et Beaudouin-Lafon, 2006]
Caroline Appert, Michel Beaudouin-Lafon. *SMCanvas : augmenter la boîte à outils Java Swing pour prototyper des techniques d'interaction avancées*. In IHM '06 : Proceedings of the 18th International Conferenceof the Association Francophone d'Interaction Homme-Machine, pages 99–106. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-350-6.
doi : http ://doi.acm.org/10.1145/1132736.1132749. Cité page 143.

[Apple Computer, 1992]
Apple Computer. *Inside Macintosh : Macintosh Toolbox Essentials*. Apple Technical Library. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1992. ISBN 0201632438. Cité page 81.

[Babar et al., 2004]
Muhammad Ali Babar, Liming Zhu, Ross Jeffery. *A Framework for Classifying and Comparing Software Architecture Evaluation Methods*. In ASWEC '04 : Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04), page 309. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2089-8. Cité page 168.

[Balakrishnan et al., 1997]
Ravin Balakrishnan, Thomas Baudel, Gordon Kurtenbach, George Fitzmaurice. *The Rockin'Mouse : integral 3D manipulation on a plane*. In CHI'97 : Proceedings of the SIGCHI conference on Human factors in computing systems, pages 311–318. ACM Press, New York, NY, USA, 1997. ISBN 0-89791-802-9.
doi : 10.1145/258549.258778. Cité page 21.

[Bass et al., 2003]
Len Bass, Paul Clements, Rick Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional, April 2003. ISBN 0321154959. Cité page 64.



[Bérard, 1999a]
François Bérard. *The Perceptual Window : Head Motion as a new Input Stream*. In Proceedings of the seventh IFIP conference on Human-Computer Interaction (INTERACT), pages 238–244. 1999a. Cité pages 53 et 156.

[Bérard, 1999b]
—. *Vision par ordinateur pour l'interaction homme-machine fortement couplée*. Thèse de Doctorat, Université Joseph Fourier, 1999b. Cité page 24.

[Bérard, 2003]
—. *The MagicTable : Computer-Vision Based Augmentation of a Whiteboard for Creative Meetings*. In Proceedings of the First ICCV Workshop on Projector-Camera Systems (PROCAMS'03). IEEE, oct 2003. Cité pages 31, 47, 60, 144 et 147.

[Bérard, 2006]
—. *The GML canvas : Aiming at Ease of Use, Compactness and Flexibility in a Graphical Toolkit*. Rapport technique TR-IMAG-CLIPS-IIHM-200601, HCI group, CLIPS-IMAG Laboratory, jan 2006. Cité page 114.

[Bérard et Coutaz, 1996]
François Bérard, Joëlle Coutaz. *Coopération de Techniques Sensorielles pour une Interaction Écologique*. In Actes des 8ème journées sur l'Interaction Homme-Machine, Grenoble, France. 1996. Cité page 137.

[Borkowski, 2006]
Stanislaw Borkowski. *Steerable Interfaces for Interactive Environments*. Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2006. Cité pages 79, 143, 150, 160 et 164.

[Borkowski et Letessier, 2006]
Stanislaw Borkowski, Julien Letessier. *User-Centric Design of a Vision System for Interactive Applications*. In ICVS'06 : Proceedings of the IEEE International Conference on Computer Vision Systems, pages 9–16. IEEE Computer Society Press, January 2006. Cité pages 79, 118, 132 et 177.

[Borkowski et al., 2005]
Stanislaw Borkowski, Julien Letessier, James L. Crowley. *Spatial Control of Interactive Surfaces in a Augmented Environment*. In Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS, volume 3245, pages 228–244. Springer, 2005. Cité pages 51, 52, 132 et 146.

[Borkowski et al., 2006]
Stanislaw Borkowski, Jérôme Maisonnasse, Julien Letessier, James L. Crowley. *Exploiter des interfaces mobiles dans le cadre d'un travail collaboratif co-présent*. In Proc. 18ème conférence francophone sur les interactions humain-machine, pages 283–284. ACM Press, apr 2006. Cité pages 29 et 153.

[Borkowski et al., 2003]
Stanislaw Borkowski, Olivier Riff, James L. Crowley. *Projecting Rectified Images in an Augmented Environment*. In Proceedings of the First ICCV Workshop on Projector-Camera Systems (PROCAMS'03). IEEE, oct 2003. Cité pages 109 et 195.

[Borkowski et al., 2004]
Stanislaw Borkowski, Sherif Sabry, James L. Crowley. *Projector-camera pair : an universal IO device for Human Machine Interaction*. In Polish National Robotics Conference KKR VIII. jun 2004. Cité page 79.

[Bradski, 2006]
Gary Bradski. *OpenCV Library Website*. 2006. Cité page 68.

[Bradski, 1998]
Gary R. Bradski. *Computer Vision Face Tracking For Use in a Perceptual User Interface*. Intel Technology Journal, (Q2) :page 15, 1998. Cité page 136.

[Bradski et Pisarevsky, 2000]
Gary R. Bradski, Vadim Pisarevsky. *Intel's Computer Vision Library : Applications in Calibration, Stereo, Segmentation, Tracking, Gesture, Face and Object Recognition*. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 2, page 2796. IEEE Computer Society Press, 2000. doi : 10.1109/CVPR.2000.854964. Cité pages 31 et 68.

[Brumitt et al., 2000]
Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, Steven Shafer. *EasyLiving : Technologies for Intelligent Environments*. In Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing (HUC'2000). sep 2000. Cité page 76.

[Buck et al., 2004]
Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, Pat Hanrahan. *Brook for GPUs : stream computing on graphics hardware*. ACM Trans. Graph., volume 23(3) :pages 777–786, 2004. ISSN 0730-0301.
doi : 10.1145/1015706.1015800. Cité page 188.

[Calaway, 2006]
Jack Calaway. *Museum of Early Video Editing Equipment and Techniques*. 2006. Cité page 27.

[Camarata et al., 2002]
Ken Camarata, Ellen Yi-Luen Do, Brian R. Johnson, Mark D. Gross. *Navigational blocks : navigating information space with tangible media*. In IUI'02 : Proceedings of the 7th international conference on Intelligent user interfaces, pages 31–38. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-459-2.
doi : 10.1145/502716.502725. Cité page 41.

[Canny, 2006]
John Canny. *The future of human-computer interaction*. ACM Queue, volume 4(6) :pages 24–32, 2006. ISSN 1542-7730.
doi : 10.1145/1147518.1147530. Cité page 171.

[Card et al., 1983]
S. Card, T. Moran, A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983. ISBN 0898598591. Cité pages 15, 41 et 56.

[Cheshire et Krochmal, 2006]
Stuart Cheshire, Marc Krochmal. *DNS-Based Service Discovery*, 2006. IETF Standard Draft. Cité page 120.

[Chrissis et al., 2003]
Mary Beth Chrissis, Mike Konrad, Sandy Shrum. *CMMI Guidlines for Process Integration and Product Improvement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 0321154967. Cité page 143.

[Chung, 1991]
Lawerence Chung. *Representation and utilization of non-functional requirements for information system design*. In CAiSE '91 : Proceedings of the third international conference on Advanced information systems engineering, pages 5–30. Springer-Verlag New York, Inc., New York, NY, USA, 1991. ISBN 0-387-54059-8. Cité page 33.

[Codehaus, 2007]
Codehaus. *The Mule open source ESB and integration platform*. 2007. Cité page 119.

[Criminisi et al., 1997]
A. Criminisi, I. Reid, A. Zisserman. *A plane measuring device*. 1997. Cité page 138.

[Crowley et Berard, 1997]
James L. Crowley, Francois Berard. *Multi-Modal Tracking of Faces for Video Communications*. In CVPR '97 : Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), page 640. IEEE Computer Society, Washington, DC, USA, 1997. ISBN 0-8186-7822-4. Cité page 136.

[Desurmont et al., 2005]
Xavier Desurmont, Bruno Lienard, Jerome Meessen, Jean-Francois Delaigle. *Real-time optimizations for integrated smart network camera*. In Nasser Kehtarnavaz, Phillip A. Laplante, editeurs, Proceedings of the International Society for Optical Engineering (SPIE) : Real-Time Imaging IX, volume 5671, pages 85–92. feb 2005.
doi : 10.1117/12.595103. Cité page 116.

[Diaz-Marino et al., 2003]
Roberto Arturo Diaz-Marino, Edward Tse, Saul Greenberg. *Programming for Multiple Touches and Multiple Users : A Toolkit for the DiamondTouch Hardware*. In Companion Proceedings of ACM UIST'03 Conference on User Interface Software and Technology. 2003. Cité page 167.

[Dick, 1985]
Philip Kindred Dick. *I Hope I Shall Arrive Soon*. Doubleday and Co. Inc., 1985. ISBN 0385195672. Cité page 15.

[Dietz et Leigh, 2001]
Paul Dietz, Darren Leigh. *DiamondTouch : a multi-user touch technology*. In UIST '01 : Proceedings of the 14th annual ACM symposium on User interface software and technology, pages 219–226. ACM Press, New York, NY, USA, 2001. ISBN 1-58113-438-X.
doi : 10.1145/502348.502389. Cité pages 21, 45 et 167.

[Digitable, 2007]
Digitable. *DIGITABLE, a shared interface for a collaborative surface*. 2007. Cité page 157.

[du Toit et Marier, 2006]
Stefanus du Toit, François Marier. *The Sh embedded embedded metaprogramming language*. 2006. Cité page 188.

[Duda et Hart, 1972]
Richard O. Duda, Peter E. Hart. *Use of the Hough transformation to detect lines and curves in pictures*. Commun. ACM, volume 15(1) :pages 11–15, 1972. ISSN 0001-0782.
doi : 10.1145/361237.361242. Cité page 68.

[Economopoulos et Martakos, 2001]
Aris Economopoulos, Drakoulis Martakos. *Component-Based Architectures For Computer Vision Systems*. In Proc. International Conference on Computer Graphics, Visualization and Computer Vision (WSCG'01). 2001. Cité page 73.

[Edwards, 1988]
A. D. N. Edwards. *The design of auditory interfaces for visually disabled users*. In CHI '88 : Proceedings of the SIGCHI conference on Human factors in computing systems, pages 83–88. ACM Press, New York, NY, USA, 1988. ISBN 0-201-14237-6.
doi : 10.1145/57167.57180. Cité page 37.

[Eisenstein et Mackay, 2006]
Jacob Eisenstein, Wendy E. Mackay. *Interacting with communication appliances : an evaluation of two computer vision-based selection techniques*. In CHI'06 : Proceedings of the SIGCHI conference on Human Factors in computing systems, pages 1111–1114. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-372-7.
doi : 10.1145/1124772.1124938. Cité page 143.

[Emonet et al., 2006]
Rémi Emonet, Dominique Vaufreydaz, Patrick Reignier, Julien Letessier. *O3MiSCID : an Object Oriented Opensource Middleware for Service Connection, Introspection and Discovery*. In 1st IEEE International Workshop on Services Integration in Pervasive Environments. June 2006. Cité pages 118, 122 et 177.

[Engelbart et al., 1967]
Douglas C. Engelbart, William K. English, Melvyn L. Berman. *Display selection techniques for text manipulation*. In IEEE Transactions on Human Factors in Electronics (HFE-8), pages 5–15. mar 1967. Cité page 20.

[EPITA, 2005]
EPITA. *Olena, a generic library for image processing*. 2005. Cité page 187.

[Fails et Olsen Jr., 2002]
Jerry Alan Fails, Dan Olsen Jr. *Light widgets : interacting in every-day spaces*. In IUI'02 : Proceedings of the 7th international conference on Intelligent user interfaces, pages 63–69. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-459-2.
doi : 10.1145/502716.502729. Cité pages 51 et 158.

[Fiala, 2005]
Mark Fiala. *ARTag, a Fiducial Marker System Using Digital Techniques*. In CVPR '05 : Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2, pages 590–596. IEEE Computer Society, Washington, DC, USA, 2005. ISBN 0-7695-2372-2.
doi : http ://dx.doi.org/10.1109/CVPR.2005.74. Cité pages 109, 127, 141 et 173.

[Fitzmaurice, 1996]
George W. Fitzmaurice. *Graspable User Interfaces*. Thèse de Doctorat, University of Toronto, Department of Computer Science, 1996. Cité page 40.

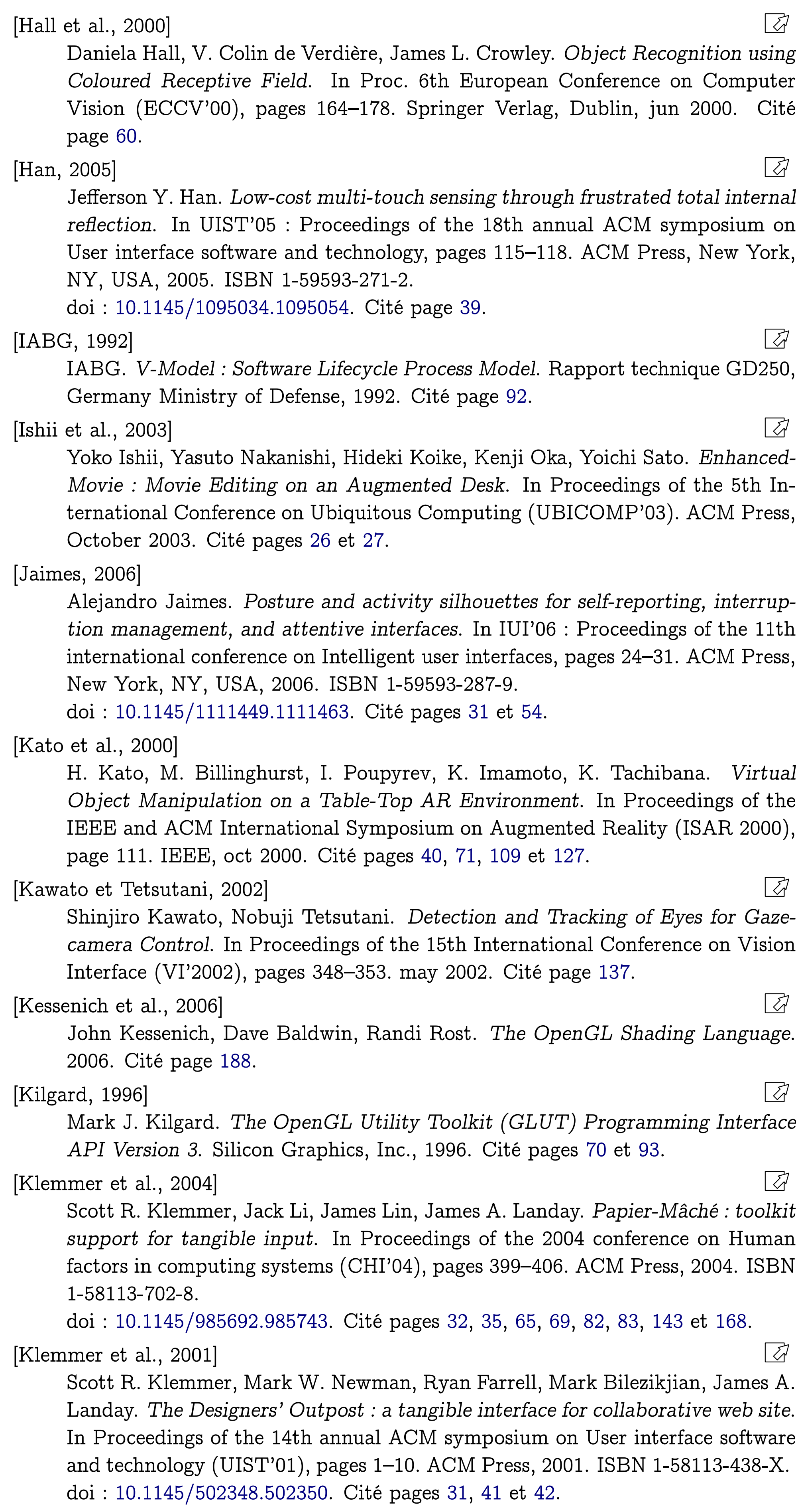
[Froehlich et al., 2006]
Bernd Froehlich, Jan Hochstrate, Verena Skuk, Anke Huckauf. *The GlobeFish and the GlobeMouse : two new six degree of freedom input devices for graphics applications*. In CHI'06 : Proceedings of the SIGCHI conference on Human Factors in computing systems, pages 191–199. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-372-7.
doi : 10.1145/1124772.1124802. Cité page 21.

[Gamma et al., 1995]
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995. Cité page 72.

[Garlan et al., 1995]
David Garlan, Robert Allen, John Ockerbloom. *Architectural mismatch or why it's hard to build systems out of existing parts*. In ICSE '95 : Proceedings of the 17th international conference on Software engineering, pages 179–185. ACM Press, New York, NY, USA, 1995. ISBN 0-89791-708-1.
doi : http ://doi.acm.org/10.1145/225014.225031. Cité page 168.

[Goldberg et Robson, 1983]
Adele Goldberg, David Robson. *Smalltalk-80 : the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983. ISBN 0-201-11371-6. Cité page 94.

[Grauman et al., 2001]
Kristen Grauman, Margrit Betke, James Gips, Gary R. Bradski. *Communication via Eye Blinks - Detection and Duration Analysis in Real Time*. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01), volume 1, page 1010. 2001. Cité page 137.



[Hall et al., 2000]
Daniela Hall, V. Colin de Verdière, James L. Crowley. *Object Recognition using Coloured Receptive Field*. In Proc. 6th European Conference on Computer Vision (ECCV'00), pages 164–178. Springer Verlag, Dublin, jun 2000. Cité page 60.

[Han, 2005]
Jefferson Y. Han. *Low-cost multi-touch sensing through frustrated total internal reflection*. In UIST'05 : Proceedings of the 18th annual ACM symposium on User interface software and technology, pages 115–118. ACM Press, New York, NY, USA, 2005. ISBN 1-59593-271-2.
doi : 10.1145/1095034.1095054. Cité page 39.

[IABG, 1992]
IABG. *V-Model : Software Lifecycle Process Model*. Rapport technique GD250, Germany Ministry of Defense, 1992. Cité page 92.

[Ishii et al., 2003]
Yoko Ishii, Yasuto Nakanishi, Hideki Koike, Kenji Oka, Yoichi Sato. *Enhanced-Movie : Movie Editing on an Augmented Desk*. In Proceedings of the 5th International Conference on Ubiquitous Computing (UBICOMP'03). ACM Press, October 2003. Cité pages 26 et 27.

[Jaimes, 2006]
Alejandro Jaimes. *Posture and activity silhouettes for self-reporting, interruption management, and attentive interfaces*. In IUI'06 : Proceedings of the 11th international conference on Intelligent user interfaces, pages 24–31. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-287-9.
doi : 10.1145/1111449.1111463. Cité pages 31 et 54.

[Kato et al., 2000]
H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, K. Tachibana. *Virtual Object Manipulation on a Table-Top AR Environment*. In Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR 2000), page 111. IEEE, oct 2000. Cité pages 40, 71, 109 et 127.

[Kawato et Tetsutani, 2002]
Shinjiro Kawato, Nobuji Tetsutani. *Detection and Tracking of Eyes for Gaze-camera Control*. In Proceedings of the 15th International Conference on Vision Interface (VI'2002), pages 348–353. may 2002. Cité page 137.

[Kessenich et al., 2006]
John Kessenich, Dave Baldwin, Randi Rost. *The OpenGL Shading Language*. 2006. Cité page 188.

[Kilgard, 1996]
Mark J. Kilgard. *The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3*. Silicon Graphics, Inc., 1996. Cité pages 70 et 93.

[Klemmer et al., 2004]
Scott R. Klemmer, Jack Li, James Lin, James A. Landay. *Papier-Mâché : toolkit support for tangible input*. In Proceedings of the 2004 conference on Human factors in computing systems (CHI'04), pages 399–406. ACM Press, 2004. ISBN 1-58113-702-8.
doi : 10.1145/985692.985743. Cité pages 32, 35, 65, 69, 82, 83, 143 et 168.

[Klemmer et al., 2001]
Scott R. Klemmer, Mark W. Newman, Ryan Farrell, Mark Bilezikjian, James A. Landay. *The Designers' Outpost : a tangible interface for collaborative web site*. In Proceedings of the 14th annual ACM symposium on User interface software and technology (UIST'01), pages 1–10. ACM Press, 2001. ISBN 1-58113-438-X.
doi : 10.1145/502348.502350. Cité pages 31, 41 et 42.

[Konrad et al., 2003]
Tollmar Konrad, David Demirdjian, Trevor Darrell. *Gesture + play : full-body*

interaction for virtual environments. In CHI'03 : CHI'03 extended abstracts on Human factors in computing systems, pages 620–621. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-637-4.
doi : 10.1145/765891.765894. Cité page 31.

[Köthe, 2000]
Ullrich Köthe. *STL-Style Generic Programming with Images*. C++ Report Magazine, volume 12(1) :pages 24–30, jan 2000. Cité page 187.

[Krueger et al., 1985]
Myron W. Krueger, Thomas Gionfriddo, Katrin Hinrichsen. *VIDEOPLACE — an artificial reality*. In CHI'85 : Proceedings of the SIGCHI conference on Human factors in computing systems, pages 35–40. ACM Press, New York, NY, USA, 1985. ISBN 0-89791-149-0.
doi : 10.1145/317456.317463. Cité pages 15, 54, 80 et 107.

[Lawson et al., 1979]
C. L. Lawson, R. J. Hanson, D. R. Kincaid, F. T. Krogh. *Basic Linear Algebra Subprograms for Fortran Usage*. ACM Trans. Math. Softw., volume 5(3) :pages 308–323, 1979. ISSN 0098-3500.
doi : 10.1145/355841.355847. Cité page 88.

[Lee et al., 2005]
Johnny C. Lee, Scott E. Hudson, Jay W. Summet, Paul H. Dietz. *Moveable interactive projected displays using projector based tracking*. In UIST'05 : Proceedings of the 18th annual ACM symposium on User interface software and technology, pages 63–72. ACM Press, New York, NY, USA, 2005. ISBN 1-59593-271-2.
doi : 10.1145/1095034.1095045. Cité pages 52 et 53.

[Letessier, 2003]
Julien Letessier. *Suivi de doigts nus pour surfaces interactives en vision par ordinateur*. Rapport de Master, Institut National Polytechnique de Grenoble, 2003. Cité pages 44, 110, 127, 128 et 148.

[Letessier et Bérard, 2004]
Julien Letessier, François Bérard. *Visual tracking of bare fingers for interactive surfaces*. In UIST'04 : Proceedings of the 17th annual ACM symposium on User interface software and technology, pages 119–122. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-957-8.
doi : 10.1145/1029632.1029652. Cité pages 44, 127, 148, 159 et 160.

[Liang et al., 1991]
J. Liang, C. Shaw, M. Green. *On Temporal-Spatial Realism in the Virtual Reality Environment*. Proceedings ACM UIST'91 4th Annual ACM Symposium on User Interface Software and Technology, pages 19–25, 1991. Cité page 164.

[LogicBlaze, 2006]
LogicBlaze. *Apache ServiceMix, the Agile open source ESB*. 2006. Cité page 119.

[Lowe, 1999]
David G. Lowe. *Object Recognition from Local Scale-Invariant Features*. In Proc. of the International Conference on Computer Vision ICCV, Corfu, pages 1150–1157. 1999. Cité page 174.

[Lux, 2004]
Augustin Lux. *The Imalab method for vision systems*. Journal of Machine Vision Applications, volume 16(1) :pages 21–26, 2004. ISSN 0932-8092.
doi : 10.1007/s00138-004-0153-6. Cité page 70.

[Mackay et Davenport, 1989]
Wendy E. Mackay, Glorianna Davenport. *Virtual video editing in interactive multimedia applications*. Commun. ACM, volume 32(7) :pages 802–810, 1989.

ISSN 0001-0782.
doi : 10.1145/65445.65447. Cité pages 22 et 27.

[MacKenzie et al., 2006]
C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, Rebekah Metz. *Reference Model for Service Oriented Architecture, Commitee Specification 1*. OASIS Open Consortium Standard, August 2006. Cité pages 64 et 74.

[Malik et Laszlo, 2004]
Shahzad Malik, Joe Laszlo. *Visual touchpad : a two-handed gestural input device*. In ICMI'04 : Proceedings of the 6th international conference on Multimodal interfaces, pages 289–296. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-995-0.
doi : 10.1145/1027933.1027980. Cité page 44.

[Matthews et al., 2004]
Tara Matthews, Anind K. Dey, Jennifer Mankoff, Scott Carter, Tye Rattenbury. *A Toolkit for Managing User Attention in Peripheral Displays*. In Proceedings of ACM UIST'04 Conference on User Interface Software and Technology. 2004.
doi : 10.1145/1029632.1029676. Cité pages 75, 83 et 195.

[Melchior et Smart, 2004]
Nik A. Melchior, William D. Smart. *A Framework for Robust Mobile Robot Systems*. In Douglas W. Gage, editeur, Proceedings of the International Society for Optical Engineering (SPIE) : Mobile Robots XVII, volume 5609, pages 145–154. December 2004. Cité page 77.

[Messerschmitt et Szyperski, 2003]
D.G. Messerschmitt, C. Szyperski. *Software Ecosystem : Understanding an Indispensable Technology and Industry*. MIT Press, 2003. Cité page 72.

[Meyers, 2004]
Scott Meyers. *The Most Important Design Guideline?* IEEE Softw., volume 21(4) :pages 14–16, 2004. ISSN 0740-7459.
doi : 10.1109/MS.2004.29. Cité page 87.

[Microsoft, 2006]
Microsoft. *Windows Communication Foundation*. 2006. Cité page 119.

[Moran et al., 1997]
Thomas P. Moran, Patrick Chiu, William van Melle. *Pen-based interaction techniques for organizing material on an electronic whiteboard*. In UIST '97 : Proceedings of the 10th annual ACM symposium on User interface software and technology, pages 45–54. ACM Press, New York, NY, USA, 1997. ISBN 0-89791-881-9.
doi : http ://doi.acm.org/10.1145/263407.263508. Cité page 161.

[Morency et Darrell, 2006]
Louis-Philippe Morency, Trevor Darrell. *Head gesture recognition in intelligent interfaces : the role of context in improving recognition*. In IUI'06 : Proceedings of the 11th international conference on Intelligent user interfaces, pages 32–38. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-287-9.
doi : 10.1145/1111449.1111464. Cité page 53.

[Morrison, 2005]
Gerald D. Morrison. *A Camera-Based Input Device for Large Interactive Displays*. IEEE Computer Graphics Applications, volume 25(4) :pages 52–57, 2005. ISSN 0272-1716.
doi : 10.1109/MCG.2005.72. Cité page 38.

[Myers et al., 2000]
Brad Myers, Scott E. Hudson, Randy Pausch. *Past, present, and future of user interface software tools*. ACM Transactions Computer-Human Interaction,

volume 7(1) :pages 3–28, 2000. ISSN 1073-0516.
doi : 10.1145/344949.344959. Cité pages 65 et 93.

[Nielsen, 1994]
J. Nielsen. *Usability Engineering*. Morgan Kaufmann, 1994. Cité page 32.

[Nielson et Olsen, 1987]
Gregory M. Nielson, Dan R. Olsen. *Direct manipulation techniques for 3D objects using 2D locator devices*. In SI3D'86 : Proceedings of the 1986 workshop on Interactive 3D graphics, pages 175–182. ACM Press, New York, NY, USA, 1987. ISBN 0-89791-228-4.
doi : 10.1145/319120.319134. Cité page 21.

[NVIDIA, 2006]
NVIDIA. *The Cg Toolkit : a high-level shading language and compiler*. 2006. Cité page 188.

[OASIS, 2006]
OASIS. *OASIS Global E-Business Consortium*. 2006. Cité page 74.

[Ohno et al., 2003]
Takehiko Ohno, Naoki Mukawa, Shinjiro Kawato. *Just blink your eyes : a head-free gaze tracking system*. In CHI'03 : CHI'03 extended abstracts on Human factors in computing systems, pages 950–957. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-637-4.
doi : 10.1145/765891.766088. Cité page 137.

[Oliver et al., 2000]
Nuria M. Oliver, Barbara Rosario, Alex Pentland. *A Bayesian Computer Vision System for Modeling Human Interactions*. IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 22(8) :pages 831–843, 2000. Cité page 173.

[Ousterhout, 1994]
John K. Ousterhout. *Tcl and the Tk toolkit*. Addison-Wesley, 1994. ISBN 020163337X. Cité pages 93 et 94.

[Owens et al., 2007]
John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, Timothy J. Purcell. *A Survey of General-Purpose Computation on Graphics Hardware*. Computer Graphics Forum, volume 26, 2007. Cité page 188.

[Pane, 2002]
John F. Pane. *HANDS : A Programming System for Children that is Designed for Usability*. Thèse de Doctorat, Carnegie Mellon University, Computer Science Department, CMU-CS-02-127, Pittsburgh, PA, may 2002. Cité page 35.

[Perry et Wolf, 1992]
Dewayne E. Perry, Alexander L. Wolf. *Foundations for the study of software architecture*. SIGSOFT Softw. Eng. Notes, volume 17(4) :pages 40–52, 1992. ISSN 0163-5948.
doi : 10.1145/141874.141884. Cité pages 63 et 87.

[Piccardi, 2004]
M. Piccardi. *Background subtraction techniques : a review*. In IEEE International Conference on Systems, Man and Cybernetics, volume 4, pages 3099–3104. 2004. ISBN 1062-922X. Cité page 128.

[Pope et Lowe, 1994]
Arthur R. Pope, David G. Lowe. *Vista : A Software Environment for Computer Vision Research*. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94). IEEE Computer Society Press, 1994.
doi : 10.1109/CVPR.1994.323895. Cité page 72.

[R. Kauth, 1977]
G. Thomas R. Kauth, A. Pentland. *BLOB : An Unsupervised Clustering Approach to Spatial Preprocessing of MSS Imagery*. In Proceedings of the 11th International Symp. Remote Sensing of the Environment. 1977. Cité page 98.

[Reignier et al., 2006]
Patrick Reignier, Sofia Zaidenberg, Rémi Emonet, Dominique Vaufreydaz, Julien Letesssier. *jOMiSCID, un intergiciel sous OSGi pour l'informatique ubiquitaire*, 2006. Atelier de travail OSGi 2006. Cité pages 122, 158 et 177.

[Rekimoto, 2002]
Jun Rekimoto. *SmartSkin : an infrastructure for freehand manipulation on interactive surfaces*. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 113–120. ACM Press, 2002. ISBN 1-58113-453-3.
doi : 10.1145/503376.503397. Cité pages 44 et 167.

[Renevier, 2004]
Philippe Renevier. *Systèmes Mixtes Collaboratifs sur Supports Mobiles : Conception et Réalisation*. Thèse de Doctorat, Université Joseph Fourier, 2004. Cité page 38.

[Ringel et al., 2004]
Meredith Ringel, Kathy Ryall, Chia Shen, Clifton Forlines, Frederic Vernier. *Release, Relocate, Reorient, Resize : Fluid Techniques for Document Sharing on Multi-User Interactive Tables*. In ACM Press, editeur, ACM Conference on Human Factors in Computing Systems (CHI'04), pages 1441–1444. apr 2004. Cité pages 29 et 43.

[Ritter, 1998]
David Ritter. *The middleware muddle*. SIGMOD Rec., volume 27(4) :pages 86–93, 1998. ISSN 0163-5808.
doi : http ://doi.acm.org/10.1145/306101.306141. Cité page 119.

[Roeber et al., 2003]
Helena Roeber, John Bacus, Carlo Tomasi. *Typing in thin air : the canesta projection keyboard - a new method of interaction with electronic devices*. In CHI'03 : CHI'03 extended abstracts on Human factors in computing systems, pages 712–713. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-637-4.
doi : 10.1145/765891.765944. Cité page 167.

[Rose, 2001]
M. Rose. *On the Design of Application Protocols*. RFC 3117, The Internet Society (IETF Network Working Group), 2001. Cité page 120.

[RWTH, 2005]
RWTH. *LTI-lib image processing and computer vision library*. 2005. Cité page 187.

[Sarin, 1996]
Sunil K. Sarin. *Object-Oriented Workflow Technology in InConcert*. compcon, volume 00 :page 446, 1996. ISSN 1063-6390.
doi : 10.1109/CMPCON.1996.501809. Cité page 77.

[Sassen et Macmillan, 2005]
Anne-Marie Sassen, Charles Macmillan. *The service engineering area : An overview of its current state and a vision of its future*. Rapport technique, European Commision, Information Society and Media Directorate-General, Software Technologies, 2005. Cité page 64.

[Scheifler et Gettys, 1986]
Robert W. Scheifler, Jim Gettys. *The X window system*. ACM Trans. Graph., volume 5(2) :pages 79–109, 1986. ISSN 0730-0301.
doi : 10.1145/22949.24053. Cité page 81.

[Senior et al., 2002]
Andrew Senior, Rein-Lien Hsu, Mohamed Abdel Mottaleb, Anil K. Jain. *Face*

Detection in Color Images. IEEE Trans. Pattern Anal. Mach. Intell., volume 24(5) :pages 696–706, 2002. ISSN 0162-8828.
doi : 10.1109/34.1000242. Cité page 31.

[Sezgin et Sankur, 2004]
Mehmet Sezgin, Bülent Sankur. *Survey over image thresholding techniques and quantitative performance evaluation*. Journal of Electronic Imaging, volume 13 :pages 146–165, janvier 2004.
doi : 10.1117/12.565754. Cité page 128.

[Shackel, 1991]
Brian Shackel. *Usability—context, framework, definition, design and evaluation*. Human factors for informatics usability, pages 21–37, 1991. Cité page 32.

[Shen et al., 2004]
Chia Shen, Frédéric D. Vernier, Clifton Forlines, Meredith Ringel. *DiamondSpin : an extensible toolkit for around-the-table interaction*. In Proceedings of the 2004 conference on Human factors in computing systems, pages 167–174. ACM Press, 2004. ISBN 1-58113-702-8.
doi : 10.1145/985692.985714. Cité page 43.

[Shneiderman, 1983]
Ben Shneiderman. *Direct Manipulation : a step beyond programming languages*. IEEE Computer, volume 16(8) :pages 57–69, aug 1983. Cité pages 22 et 23.

[Shneiderman, 1986]
—. *Empirical studies of programmers : the territory, paths, and destination*. In Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers, pages 1–12. Ablex Publishing Corp., Norwood, NJ, USA, 1986. ISBN 0-89391-388-X. Cité page 117.

[Smith et Williams, 2000]
Connie U. Smith, Lloyd G. Williams. *Software performance antipatterns*. In WOSP '00 : Proceedings of the 2nd international workshop on Software and performance, pages 127–136. ACM Press, New York, NY, USA, 2000. ISBN 1-58113-195-X.
doi : 10.1145/350391.350420. Cité page 95.

[Streitz et al., 1999]
Norbert A. Streitz, Jörg Geißler, Torsten Holmer, Shin'ichi Konomi, Christian Müller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, Ralf Steinmetz. *i-LAND : an interactive landscape for creativity and innovation*. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 120–127. ACM Press, 1999. ISBN 0-201-48559-1.
doi : 10.1145/302979.303010. Cité page 51.

[Sugimoto et al., 2001]
M. Sugimoto, F. Kusunoki, H. Hashizume. *E2Board : An Electronically Enhanced Board for Games and Group Activity Support*. In Proceedings of the International Conference on Affective Human Factors Design. Singapore, jun 2001. Cité page 22.

[Sugimoto et al., 2004]
Masanori Sugimoto, Kazuhiro Hosoi, Hiromichi Hashizume. *Caretta : a system for supporting face-to-face collaboration by integrating personal and shared spaces*. In Proceedings of the 2004 conference on Human factors in computing systems, pages 41–48. ACM Press, 2004. ISBN 1-58113-702-8.
doi : 10.1145/985692.985698. Cité pages 22, 46 et 47.

[Sukaviriya et al., 2003]
Noi Sukaviriya, Mark Podlaseck, Rick Kjeldsen, Anthony Levas, Gopal Pingali, Claudio Pinhanez. *Embedding Interactions in a Retail Store Environment : The Design and Lessons Learned*. In Proceedings of the Ninth IFIP International

Conference on Human-Computer Interaction (INTERACT'03). 2003. Cité pages 49 et 50.

[Sukthankar et al., 2000]
Rahul Sukthankar, Robert G. Stockton, Matthew D. Mullin. *Automatic Keystone Correction for Camera-Assisted Presentation Interfaces*. In ICMI '00 : Proceedings of the Third International Conference on Advances in Multimodal Interfaces, pages 607–614. Springer-Verlag, London, UK, 2000. ISBN 3-540-41180-1. Cité page 139.

[Takao et al., 2003]
Naoya Takao, Jianbo Shi, Simon Baker. *Tele-Graffiti : A Camera-Projector Based Remote Sketching System with Hand-Based User Interface and Automatic Session Summarization*. International Journal of Computer Vision, volume 53(2) :pages 115 – 133, July 2003. Cité page 26.

[Tomasi et Kanade, 1991]
Carlo Tomasi, Takeo Kanade. *Detection and Tracking of Point Features*. Rapport technique CMU-CS-91-132, Carnegie Mellon University, April 1991. Cité page 136.

[Turk, 2004]
Matthew Turk. *Computer vision in the interface*. Commun. ACM, volume 47(1) :pages 60–67, 2004. ISSN 0001-0782.
doi : 10.1145/962081.962107. Cité page 19.

[Ullmer et Ishii, 1997]
Brygg Ullmer, Hiroshi Ishii. *The metaDESK : models and prototypes for tangible user interfaces*. In Proceedings of the 10th annual ACM symposium on User interface software and technology (UIST'97), pages 223–232. ACM Press, 1997. ISBN 0-89791-881-9.
doi : 10.1145/263407.263551. Cité page 26.

[Underkoffler et Ishii, 1998]
John Underkoffler, Hiroshi Ishii. *Illuminating light : an optical design tool with a luminous-tangible interface*. In CHI'98 : Proceedings of the SIGCHI conference on Human factors in computing systems, pages 542–549. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998. ISBN 0-201-30987-4.
doi : 10.1145/274644.274717. Cité page 23.

[van Rossum, 2006]
Guido van Rossum. *The Python Programming Language*. 2006. Cité page 94.

[Vinoski, 1997]
S. Vinoski. *CORBA : integrating diverse applications within distributed heterogeneous environments*. Communications Magazine, IEEE, volume 35(2) :pages 46–55, 1997. Cité page 74.

[Vogel et Balakrishnan, 2004]
Daniel Vogel, Ravin Balakrishnan. *Interactive public ambient displays : transitioning from implicit to explicit, public to personal, interaction with multiple users*. In UIST '04 : Proceedings of the 17th annual ACM symposium on User interface software and technology, pages 137–146. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-957-8.
doi : 10.1145/1029632.1029656. Cité page 31.

[Vogel et Balakrishnan, 2005]
—. *Distant freehand pointing and clicking on very large, high resolution displays*. In UIST'05 : Proceedings of the 18th annual ACM symposium on User interface software and technology, pages 33–42. ACM Press, New York, NY, USA, 2005. ISBN 1-59593-271-2.
doi : 10.1145/1095034.1095041. Cité page 29.

[Wagner et Schmalstieg, 2006]
Daniel Wagner, Dieter Schmalstieg. *Handheld Augmented Reality Displays*.

IEEE Virtual Reality Conference (VR'2006), volume 0 :page 321, 2006. ISSN 1087-8270.
doi : 10.1109/VR.2006.67. Cité page 71.

[Wang et Canny, 2006]
Jingtao Wang, John Canny. *TinyMotion : camera phone based interaction methods*. In CHI '06 : extended abstracts on Human factors in computing systems, pages 339–344. ACM Press, New York, NY, USA, 2006.
doi : 10.1145/1125451.1125526. Cité page 53.

[Watson et al., 1998]
B. Watson, N. Walker, W. Ribarsky, V. Spaulding. *Effects of variation in system responsiveness on user performance in virtual environments*. Human Factors, Special Section on Virtual Environments, volume 3(40) :pages 403–414, 1998. Cité pages 56 et 130.

[Wellner, 1993a]
Pierre D. Wellner. *Adaptive Thresholding for the DigitalDesk*. Rapport technique EPC-1993-110, Rank Xerox Ltd, 1993a. Cité pages 31 et 111.

[Wellner, 1993b]
—. *Interacting with Paper on the DigitalDesk*. Communications of the ACM, volume 36(7) :pages 86–97, 1993b. Cité page 31.

[Wellner, 1993c]
—. *Self Calibration for the DigitalDesk*. Rapport technique EPC-1993-109, Rank Xerox Ltd, 1993c. Cité page 31.

[Whaley et al., 2001]
R. Clint Whaley, Antoine Petitet, Jack J. Dongarra. *Automated empirical optimizations of software and the ATLAS project*. Parallel Computing, volume 27(1–2) :pages 3–35, 2001. Cité page 89.

[Wilson, 2004]
Andrew D. Wilson. *TouchLight : an imaging touch screen and display for gesture-based interaction*. In ICMI'04 : Proceedings of the 6th international conference on Multimodal interfaces, pages 69–76. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-995-0.
doi : 10.1145/1027933.1027946. Cité pages 31 et 166.

[Wilson, 2006]
—. *Robust computer vision-based detection of pinching for one and two-handed gesture input*. In UIST '06 : Proceedings of the 19th annual ACM symposium on User interface software and technology, pages 255–258. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-313-1.
doi : http ://doi.acm.org/10.1145/1166253.1166292. Cité page 91.

[Wu et al., 2000]
Andrew Wu, Mubarak Shah, N. da Vitoria Lobo. *A Virtual 3D Blackboard : 3D Finger Tracking using a single camera*. In Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00). mar 2000. Cité pages 25, 29, 38 et 44.

[Wu et Balakrishnan, 2003]
Mike Wu, Ravin Balakrishnan. *Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays*. In Proceedings of the 16th annual ACM symposium on User interface software and technology, pages 193–202. ACM Press, 2003. ISBN 1-58113-636-6.
doi : 10.1145/964696.964718. Cité pages 21 et 45.

[Yang et al., 2004]
Jian Yang, David Zhang, Alejandro F. Frangi, Jing yu Yang. *Two-Dimensional PCA : A New Approach to Appearance-Based Face Representation and Recognition*. IEEE Trans. Pattern Anal. Mach. Intell., volume 26(1) :pages 131–137,

2004. ISSN 0162-8828.
doi : http ://dx.doi.org/10.1109/TPAMI.2004.1261097. Cité page 174.

[Yee, 2003]
Ka-Ping Yee. *Peephole displays : pen interaction on spatially aware handheld computers*. In Proceedings of the conference on Human factors in computing systems (CHI'03), pages 1–8. ACM Press, 2003. ISBN 1-58113-630-7. Cité page 51.

[Zhao et al., 2003]
W. Zhao, R. Chellappa, P. J. Phillips, A. Rosenfeld. *Face recognition : A literature survey*. ACM Comput. Surv., volume 35(4) :pages 399–458, 2003. ISSN 0360-0300.
doi : 10.1145/954339.954342. Cité pages 31 et 127.