HAL
open science

Plateforme autonomique dirigée par les modèles pour la construction d'interfaces multimodales dans les environnements pervasifs

Pierre-Alain Avouac

To cite this version:

Pierre-Alain Avouac. Plateforme autonomique dirigée par les modèles pour la construction d'interfaces multimodales dans les environnements pervasifs. Interface homme-machine [cs.HC]. Université de Grenoble, 2013. Français. NNT : 2013GRENM084 . tel-01074122

HAL Id: tel-01074122

https://theses.hal.science/tel-01074122v1

Submitted on 13 Oct 2014

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

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

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Pierre-Alain AVOUAC

Thèse dirigée par **Laurence NIGAY**
et codirigée par **Philippe LALANDA**

préparée au sein du **Laboratoire d'informatique de Grenoble (LIG)**
et de l'**École doctorale mathématiques, sciences et technologies de l'information, informatique (MSTII)**

Plateforme autonomique dirigée par les modèles pour la construction d'interfaces multimodales dans les environnements pervasifs

Thèse soutenue publiquement le **7 février 2013**,
devant le jury composé de :

Claudia RONCANCIO
professeur de Grenoble INP, présidente
Michel RIVEILL
professeur de l'université de Nice - Sophia Antipolis, rapporteur
Philippe PALANQUE
professeur de l'université Paul Sabatier, rapporteur
Maxime FARACO
directeur recherche et développement chez Schneider Electric, examinateur
Philippe ROOSE
maitre de conférences de l'université de Pau et des Pays de l'Adour, examinateur
Laurence NIGAY
professeur de l'UJF, directrice de thèse
Philippe LALANDA
professeur de l'UJF, co-directeur de thèse

Résumé

Au sein des environnements pervasifs, nous partons du constat de la multiplicité des possibilités d'interaction à notre disposition (comme les dispositifs d'interaction variés dans une maison incluant des télécommandes, des manettes de jeu, des téléphones, des tablettes, des objets augmentés) et de la multiplicité des services informatiques destinés à notre quotidien. Nos travaux répondent alors à un problème important aujourd'hui de gestion dynamique et non figée lors de la conception de l'interaction multimodale dans des environnements pervasifs, qui sont hétérogènes et dynamiques. Pour cela notre contribution conceptuelle couvre à la fois la spécification et l'exécution d'une interface multimodale en environnements pervasifs et s'articule selon trois facettes complémentaires : un langage de spécification de l'interaction multimodale, un gestionnaire autonomique et une plateforme d'intégration. Le gestionnaire autonomique est en charge de faire le lien entre les services et les dispositifs d'interaction en créant un modèle d'interaction multimodale adapté à l'état courant de l'environnement. Ce modèle est alors mis en oeuvre par notre plateforme d'intégration. Notre proposition conceptuelle est mise en oeuvre par notre plateforme logicielle DynaMo, complètement opérationnelle et stable : DynaMo exploite deux technologies au-dessus d'OSGi, iPOJO pour le modèle à composant et Cilia comme plateforme de médiation.

Summary

In pervasive environments, with the proliferation of communicating devices in the environments (e.g., remoter controller, gamepad, mobile phone, augmented object), the users will express their needs or desires to an enormous variety of services with a multitude of available interaction modalities, expecting concurrently the environment and its equipment to react accordingly. Addressing the challenge of dynamic management at runtime of multimodal interaction in pervasive environments, our contribution is dedicated to software engineering of dynamic multimodal interfaces by providing: a specification language for multimodal interaction, an autonomic manager and an integration platform. The autonomic manager uses models to generate and maintain a multimodal interaction adapted to the current conditions of the environment. The multimodal interaction data-flow from input devices to a service is then effectively realized by the integration platform. Our conceptual solution is implemented by our DynaMo platform that is fully operational and stable. DynaMo is based on iPOJO, a dynamic service-oriented component framework built on top of OSGi and on Cilia, a component-based mediation framework.

Remerciements

Ce doctorat a été une aventure marquante par son étrangeté, sa difficulté et son intensité.
Je remercie :
- les examinateurs et les rapporteurs pour leurs critiques de mon travail ;
- toutes les personnes qui m'ont aidé, m'ont soutenu ou ont nourri mes réflexions, notamment Elmehdi, Issac, Jian Qi, Matthieu et Yoann ;
- mes directeurs de thèse qui ont su me guider dans mes recherches ; merci Philippe pour ta confiance et tes encouragements, merci Laurence pour ton efficacité et ton absence d'ambage ;
- Anaïs, pour ta gentillesse, ta disponibilité, et la légerté que tu as amené dans ma vie.

Et les linuxfriens, les wikipédiens, France Culture, les créateurs de sons.

Table des matières

1 Introduction **11**
1.1 Contexte . . . 11
1.1.1 Succès des nouveaux dispositifs d'interaction . . . 12
1.1.1.1 Écran plat . . . 12
1.1.1.2 Wii Remote . . . 12
1.1.1.3 Smartphone à écran tactile . . . 13
1.2 Objectifs et contributions de la thèse . . . 14
1.3 Organisation du document . . . 15
1.3.1 Présentation du contexte . . . 15
1.3.2 Environnements pervasifs multimodaux : nos solutions conceptuelles et logicielles . . . 16

2 Informatique pervasive et multimodalité **18**
2.1 Informatique pervasive . . . 18
2.1.1 Hétérogénéité . . . 21
2.1.1.1 Protocoles de communication . . . 21
2.1.1.2 Dispositifs d'interaction . . . 22
2.1.2 Dynamisme . . . 27
2.2 Interfaces multimodales . . . 29
2.2.1 Dispositif d'interaction . . . 30
2.2.2 Langage d'interaction . . . 30
2.2.3 Modalité d'interaction . . . 31
2.2.4 Multimodalité . . . 32
2.2.4.1 Définition . . . 32
2.2.4.2 Choix de modalités : équivalence . . . 33
2.2.4.3 Combinaison de modalités : complémentarité et redondance . . . 33
2.2.4.4 Synthèse . . . 34
2.3 Conclusion : pourquoi la multimodalité en environnement pervasif ? . . . 34
2.3.1 Apport de la multimodalité pour les environnements pervasifs . . . 34
2.3.2 Greffe de multimodalité : notre hypothèse . . . 35

3 Plateformes logicielles pour l'informatique pervasive et l'interaction multimodale **37**
3.1 Plateformes généralistes pour le dynamisme . . . 37
3.1.1 Introduction . . . 38
3.1.2 OSGi . . . 40
3.1.3 iPOJO . . . 41

3.1.4 Cilia . . . 42
3.1.5 Plateformes pour des environnements pervasifs . . . 44
3.1.5.1 iROS . . . 44
3.1.5.2 Gaia . . . 45
3.1.5.3 WComp . . . 46
3.1.5.4 Smart-M3 . . . 47
3.1.6 Synthèse . . . 49
3.2 Conception d'interfaces multimodales . . . 49
3.2.1 ICon . . . 50
3.2.2 ICARE . . . 51
3.2.3 Squidy . . . 53
3.2.4 OpenInterface . . . 55
3.2.4.1 OIDE . . . 56
3.2.4.2 SKEMMI . . . 56
3.2.5 Synthèse . . . 57
3.2.5.1 Spécification de l'interface multimodale . . . 57
3.2.5.2 Intégration de l'interface dans une application interactive . . . 57
3.2.5.3 Dynamisme et adaptation automatique . . . 58
3.3 Conclusion . . . 58

4 Proposition conceptuelle : plateforme pour la multimodalité en environnement pervasif 60
4.1 Architecture conceptuelle . . . 60
4.2 Du développement à l'exécution . . . 62
4.2.1 Trois acteurs . . . 62
4.2.2 Cycle de développement . . . 62
4.2.2.1 Description et mandataire d'un dispositif ou d'une application . 63
4.2.2.2 Description d'une interaction . . . 64
4.2.2.3 Configuration de l'utilisateur . . . 65
4.3 Langage de spécification pour les interactions multimodales . . . 65
4.3.1 Composant . . . 65
4.3.2 Assemblage . . . 66
4.4 Plateforme d'intégration . . . 67
4.4.1 Plateforme à services . . . 68
4.4.2 Plateforme de médiation . . . 69
4.5 Gestionnaire autonomique . . . 70
4.5.1 Principe de fonctionnement . . . 71
4.5.2 Sélection de modèles . . . 71
4.5.2.1 Propriétés d'un ensemble de modèles . . . 72
4.5.2.2 Construction d'ensembles valides et cohérents . . . 74
4.5.2.3 Heuristique : maximisation du nombre de dispositifs . . . 75
4.5.2.4 Heuristique : maximisation de la multimodalité « conjointe » . . 76
4.5.2.5 Heuristique : minimisation de la distance sémantique . . . 76
4.5.2.6 Application des heuristiques . . . 78
4.5.2.7 Synthèse . . . 79
4.5.3 Composition et transformation de modèles . . . 80
4.5.3.1 Composition . . . 80
4.5.3.2 Transformation . . . 81

4.5.3.3 Transformation : exploitation des ports libres des dispositifs et des applications . . . 81
4.5.3.4 Transformation : adaptation des types de données et des intervalles de valeurs . . . 82
4.5.3.5 Transformation : adaptation à un dispositif universel . . . 83
4.5.3.6 Transformation en une chaine de médiation . . . 85
4.6 Synthèse . . . 85

5 DynaMo : une plateforme pour la multimodalité en environnement pervasif 88
5.1 Langage de spécification pour les interfaces multimodales . . . 89
5.1.1 À propos de la notation graphique . . . 89
5.1.2 Métamodèle de conception . . . 89
5.1.3 Métamodèles pour le programmeur . . . 92
5.1.3.1 Métamodèle des mandataires . . . 93
5.1.3.2 Métamodèle des médiateurs . . . 95
5.1.4 Métamodèle pour le gestionnaire autonomique . . . 97
5.1.4.1 Métamodèle des composants . . . 97
5.1.4.2 Métamodèle des fragments d'interaction . . . 98
5.1.5 Synthèse . . . 100
5.2 Plateforme d'intégration . . . 100
5.2.1 Plateforme à service . . . 100
5.2.1.1 Module de découverte . . . 101
5.2.1.2 Communication avec une chaine de médiation . . . 103
5.2.2 Plateforme de médiation . . . 104
5.2.3 Synthèse . . . 104
5.3 Gestionnaire autonomique . . . 105
5.3.1 Transformation : solveurs . . . 105
5.3.1.1 Solveurs par défaut . . . 105
5.3.1.2 Solveur d'extension . . . 106
5.4 Conclusion . . . 107
5.4.1 Quelques chiffres . . . 107
5.4.2 Une implémentation fonctionnelle . . . 107

6 DynaMo à l'exécution 108
6.1 Évaluation des performances techniques . . . 108
6.1.1 Temps de propagation des données . . . 108
6.1.2 Temps d'adaptation . . . 109
6.1.3 Discussion sur les performances . . . 110
6.2 Exemples développés avec DynaMo . . . 110
6.2.1 Présentation des dispositifs et des applications . . . 111
6.2.2 Génération d'interfaces : absence d'information . . . 113
6.2.2.1 Dispositif unique . . . 113
6.2.2.2 Plusieurs dispositifs . . . 113
6.2.3 Génération d'interfaces : absence de généricité . . . 114
6.2.4 Génération d'interfaces : généricité . . . 115
6.2.4.1 Dispositif unique . . . 115
6.2.4.2 Plusieurs dispositifs . . . 117
6.3 Conclusion . . . 118

7 Conclusion **122**
7.0.1 Contributions de la thèse 122
7.0.2 Perspectives de recherche 125
7.0.2.1 Peupler la plateforme en vue d'évaluations expérimentales 125
7.0.2.2 Enrichir le gestionnaire autonomique 125
7.0.2.3 Rendre observable et contrôlable les décisions du gestionnaire autonomique 126

A Métamodèles **128**

Bibliographie **132**

Table des figures

1.1 Comparaison d'écrans plat et cathodique . . . 12
1.2 Wiimote . . . 13
1.3 Structure du document . . . 15

3.1 Représentation d'un composant . . . 39
3.2 Architecture orientée service . . . 39
3.3 Cycle de vie d'un bundle OSGi . . . 40
3.4 Exemple de composant iPOJO . . . 42
3.5 Médiateur Cilia . . . 43
3.6 Exemple de chaine de médiation Cilia . . . 44
3.7 Exemple d'utilisation de Smart-M3 . . . 48
3.8 Exemple d'utilisation de composants ICARE . . . 52
3.9 Exemple d'utilisation de tubes et filtres Squidy . . . 54

4.1 Rôle du gestionnaire autonomique . . . 61
4.2 Architecture globale . . . 62
4.3 Comparaison des cycles de développement . . . 63
4.4 Emplacement des mandataires . . . 64
4.5 Exemple de composant . . . 66
4.6 Exemple d'assemblages . . . 67
4.7 Organisation de la découverte . . . 69
4.8 Exemple de chaine de médiation . . . 70
4.9 Rôle de la plateforme de médiation . . . 70
4.10 Détail du fonctionnement du gestionnaire autonomique . . . 71
4.11 Exemple de modèles d'interaction incompatibles . . . 73
4.12 Exemples d'ensembles valides, invalides, cohérents et incohérents . . . 75
4.13 Exemple de maximisation du nombre de dispositifs . . . 77
4.14 Exemples de maximisation de la multimodalité « conjointe » . . . 78
4.15 Exemples de minimisation de distance sémantique . . . 79
4.16 Exemple de groupements sémantiques . . . 83

5.1 Métamodèle des applications, dispositifs et composants . . . 90
5.2 Métamodèle des configurations . . . 91
5.3 Généralisation : les instances et les ports d'instances. . . . 91
5.4 Métamodèle des configurations . . . 91
5.5 Métamodèle des types de données . . . 93
5.6 Métamodèle des mandataires . . . 94
5.7 Métamodèle des médiateurs . . . 95

5.8 Modèle de médiateur « déclencheur » 97
5.9 Modèle de médiateur « fusion » 98
5.10 Métamodèle des composants 99
5.11 Métamodèle des fragments d'interaction 99
5.12 Module et sous-modules de découverte 101

6.1 Interface générée en l'absence de toute information 113
6.2 Autre interface générée en l'absence de toute information 114
6.3 Fragment d'interaction : KSudoku, Joystick et Bouton Rotatif 115
6.4 Interface générée en l'absence de généricité 115
6.5 Fragment d'interaction : KSudoku et Gamepad 116
6.6 Fragment d'interaction : Gamepad et BDRC 116
6.7 Interface générée à partir de deux fragments d'interaction 117
6.8 Fragment d'interaction : VLC et Media Player 118
6.9 Fragment d'interaction : Media Player et Gamepad 118
6.10 Fragment d'interaction : Media Player et Wiimote 120
6.11 Interface générée entre VLC, la BDRC et la Wiimote 121

7.1 Multimodalité et environnements pervasifs : multiplicité, flexibilité et robustesse 123
7.2 Pluridisciplinarité des travaux 123
7.3 Graphe du nombre de modèles d'interaction par application : 3 cas 124

A.1 Métamodèle des types de données 128
A.2 Métamodèle des interactions pour le concepteur 129
A.3 Métamodèles des mandataires et des médiateurs pour le programmeur 130
A.4 Métamodèle des fragments d'interaction pour le gestionnaire autonomique 131

Liste des tableaux

2.1 Taxonomie de Buxton 24

3.1 Positionnement de ICon suivant notre grille d'analyse 51
3.2 Positionnement de ICARE suivant notre grille d'analyse 53
3.3 Positionnement de Squidy suivant notre grille d'analyse 55
3.4 Positionnement de OIDE suivant notre grille d'analyse 56
3.5 Positionnement de SKEMMI suivant notre grille d'analyse 57

4.1 Compatibilité entre les types de données 84

Chapitre 1

Introduction

1.1 Contexte

Par l'automatisation du traitement de l'information, l'informatique offre de nouvelles possibilités aux humains. L'avènement des réseaux informatiques, et notamment de leur interconnexion à l'échelle de la planète, a permis d'accroitre la quantité d'informations accessibles. Ainsi de très nombreuses activités peuvent s'appuyer sur les systèmes informatiques : s'informer et partager de l'information, se cultiver, se divertir, apprendre, communiquer avec ses semblables, etc.

L'informatique évolue. Les matériels informatiques se font **plus nombreux** dans le quotidien de nombreuses personnes. Ses formes **se diversifient** : smartphone, tablette, table et tableau interactifs, liseuse, baladeur numérique, dispositif pour la navigation par satellite, vidéoprojecteur, *plug computer*, etc. Certains de ces matériels démocratisent des techniques d'interaction, notamment l'interaction tactile directement sur un écran, la reconnaissance de gestes et de parole, ainsi que la synthèse vocale. Ces techniques d'interaction ne sont pas nouvelles, mais leur **déploiement à grande échelle** est récent.

Cette évolution, tant en nombre qu'en diversité, ne pose pas de problème [1] en tant que tel : chaque matériel effectue le travail pour lequel il a été prévu. Éventuellement, des matériels **compatibles** communiquent entre eux, s'organisent pour fournir un meilleur service à l'utilisateur, plus complet, ou bien plus rapide, ou encore plus précis. Plutôt que de poser des problèmes, cette évolution offre de **nouvelles possibilités**. En effet, des matériels peuvent être employés pour des utilisations auxquels ils n'ont pas été prévus, c'est notamment le cas des dispositifs d'interaction. Cependant, le manque de compatibilité entre les dispositifs d'interaction et les services est un facteur important de limitation : quasiment tous se limitent à contrôler seulement une infime partie de ce qu'ils pourraient contrôler. Par exemple, une manette de jeu sans fil peut être employée comme télécommande domotique. L'**hétérogénéité** des protocoles de communication [2] est un facteur majeur de cette limitation. Ainsi, l'utilisateur n'est pas capable d'**exploiter pleinement** tous les matériels à sa disposition. De son côté, un industriel ne peut fournir un matériel, logiciel, ou service capable de s'**interfacer au mieux** avec le reste de l'environnement informatique de son client.

La recherche en informatique a un rôle majeur à jouer pour exploiter au mieux ces évolutions, particulièrement la recherche dans le domaine du **génie logiciel**, qui vise à faciliter la

1. L'augmentation de la consommation de matériel informatique est par contre susceptible de provoquer des problèmes sociétaux ou écologiques [United Nations Environment Programme, 2005].

2. Exemples de protocoles : Bluetooth, Bonjour, D-Bus, DCOM, RMI, UPnP, X10, ZigBee. Même si ces standards ne couvrent pas un même périmètre fonctionnel, ils visent tous à permettre l'échange d'informations.

conception et la mise en œuvre de logiciels. En effet, une approche globale semble nécessaire : comment faire pour surmonter cette absence de communication et d'organisation entre les dispositifs ? D'un autre côté, la recherche en **interaction homme-machine** (IHM) vise à faciliter la communication entre les humains et les machines. Or, une des problématiques qui émerge de la multiplication des dispositifs est de permettre à l'utilisateur de les utiliser pour communiquer avec l'ensemble de son environnement informatique. Notamment, comment un utilisateur qui est susceptible d'avoir à sa disposition **plusieurs dispositifs d'interaction**, ou **plusieurs manières d'utiliser un dispositif** peut interagir avec tous ses matériels informatiques. Ces types d'interaction sont qualifiés de multimodaux.

Les interactions multimodales supposent une multiplicité de dispositifs d'interaction, ou un seul dispositif utilisable de plusieurs manières. Actuellement, le nombre et la place des matériels informatiques évoluent fortement. Pourtant, il semble que les manières d'interagir avec ces matériels n'évoluent pas de manière aussi rapide. Par exemple, nous considérons des dispositifs d'interaction qui ont eu un certain succès commercial depuis les années 1990 et qui n'ont guère évolués depuis. Pour ces dispositifs nous passons en revue les raisons plausibles de leur succès du point de vue des interactions, et notamment l'incitation ou l'absence de barrière qui a poussé leur acquisition.

1.1.1 Succès des nouveaux dispositifs d'interaction

1.1.1.1 Écran plat

L'écran plat s'est démocratisé à la fin des années 1990, avec la technologie à cristaux liquides. Au niveau de l'interaction, un écran plat n'amène pas de différences majeures. Sa finesse peut néanmoins lui permettre d'être accroché plus facilement aux murs, ou d'être incorporé dans un matériel mobile, ce qui a pour conséquence d'augmenter le nombre de contextes d'utilisation possibles. Compatible avec la norme VGA [3], norme dominante à l'époque pour la communication entre l'ordinateur et l'écran, les écrans plats n'ont pas impliqués des modifications des systèmes d'exploitation ni des applications. Ainsi l'utilisateur a pu acquérir un écran plat sans changer le reste de son matériel, ni changer ses logiciels.

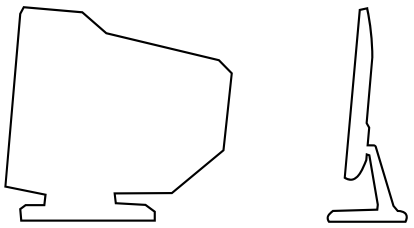


FIGURE 1.1 – Écrans vus de profil : écran cathodique à gauche, écran plat à droite.

1.1.1.2 Wii Remote

La Wii Remote, ou Wiimote, est un dispositif d'interaction d'entrée et de sortie sans fil. Elle a été commercialisée à partir de 2006 comme manette de la Wii, une console de jeu. Elle dispose de douze boutons, d'un accéléromètre fournissant trois valeurs et d'une caméra infrarouge. La caméra infrarouge s'utilise avec une barre distante qui dispose de plusieurs LED infrarouges. La Wiimote peut alors servir de dispositif de pointage. Elle possède un système

3. VGA est l'acronyme de *video graphics array*.

mécanique capable de faire vibrer la manette, d'un haut parleur, et de quatre LED. Un port de connexion est présent pour connecter un autre dispositif, permettant par exemple des interactions bi-manuelles. Elle utilise le standard de communication Bluetooth. Elle a été utilisée dans plusieurs projets de recherche [Vicaria *et al.*, 2008], notamment pour l'exploration de techniques d'interaction [Lee, 2008]. Au total, environ 87 millions de consoles Wii ont été vendues jusqu'en juin 2011 [4], chaque console étant vendue avec une manette. Une manette pouvait être achetée sans console.

FIGURE 1.2 – Wiimote.

Nintendo, l'entreprise à l'origine de la Wiimote, fournissait des kits de développement uniquement pour exploiter la Wiimote sur la console Wii. La manette pouvant néanmois communiquer avec un ordinateur via le protocole Bluetooth, des bibliothèques de récupération des données issues de la manette ont été développées [Lee, 2008]. La Wiimote a alors pu être utilisée comme télécommande pour le contrôle de lampes [5] et le contrôle de lecteurs audio [6], comme dispositif d'entrée pour la génération de son [7], etc. Ces exemples d'intégrations d'un dispositif d'interaction dans une application ont été réalisés par les utilisateurs des logiciels, dans un cadre non professionnel.

Il est clair que si la Wiimote s'est retrouvée chez beaucoup de particuliers, c'est parce qu'elle était vendue comme accessoire principale d'une console de jeu, et non pas en tant que dispositif d'interaction permettant de contrôler un environnement pervasif. D'un autre côté, le marketing pour la Wii est parfois basé sur la technique d'interaction que permet la manette, voir même uniquement dessus [8]. Cependant, le travail d'intégration démontre clairement que le besoin et l'envie existent, mais aussi qu'il manque une approche globale puisque chaque logiciel est adapté, via éventuellement une extension, pour prendre en compte la Wiimote.

1.1.1.3 Smartphone à écran tactile

Le smartphone à écran tactile est en train de se démocratiser. Au niveau de l'interaction, le pointage est plus direct qu'avec la souris. Bien souvent, un smartphone possède un accéléromètre tridimensionnel qui lui permet de connaitre ses déplacements dans l'espace ; il permet donc des techniques d'interaction basées sur le geste de la main. Bien qu'il soit plus encombrant qu'un téléphone portable classique actuel, il reste facilement transportable par une personne. Le taux d'équipement des téléphones portables [9] étant élevé en France lors de l'apparition des smarphones, ce dernier a vraisemblablement profité de l'effet de remplacement [10], de l'attrait de

4. Nintendo consolidated sales transition by region : `http://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1106.pdf`.
5. Projet Wii-lla : `http://www.opendmx.net/index.php/Wiimote_Control`.
6. Clementine : `http://code.google.com/p/clementine-player/wiki/WiiRemotes`.
7. Extension pour le logiciel SuperCollider : `http://petemoss.org/SuperCollider/`.
8. Par exemple, les spots publicitaires `http://www.youtube.com/watch?v=SAYBiRcduJk`.
9. En 2008, 79,9% des ménages dont la personne de référence a 16 ans ou plus en France métropolitaine possède au moins un téléphone portable, selon le *Tableaux de l'économie française*, réalisé par l'INSEE, édition 2012, section 6.2, page 77. `http://www.insee.fr/fr/ffc/tef/tef2012/tef2012.pdf`.
10. Les téléphones portables ont une durée de vie de moins de deux ans dans les pays industrialisés [United Nations Environment Programme, 2005].

nouvelles fonctionnalités sur un dispositif dont l'usage est répandu : en termes d'application, un smartphone est plus proche de l'ordinateur individuel que d'un téléphone portable classique. Il est difficile d'évaluer l'incitation de la nouvelle technique d'interaction. Comme pour la Wii, le marketing peut mettre en avant la technique d'interaction [11].

Enfin, au delà des raisons techniques développées ci-dessus, le marketing est probablement un facteur très important au regard des sommes mises en jeu [12].

1.2 Objectifs et contributions de la thèse

Ces travaux de recherche visent à faciliter la pénétration des interfaces multimodales chez le grand public à court et moyen termes. Concepteurs d'interaction, développeurs et utilisateurs doivent se voir proposer des solutions pour exploiter pleinement les contextes d'utilisation actuels, mais aussi ceux à venir. Ces solutions doivent permettre une mise en œuvre facile et rapide d'interfaces multimodales.

En considérant un système interactif composé d'un noyau fonctionnel et d'une interface multimodale entre ce noyau et l'utilisateur, les modalités d'interaction constituent des médiateurs matériels et logiciels entre l'utilisateur et le noyau fonctionnel. Nos travaux portent sur les modalités et la multimodalité en entrée, de l'utilisateur vers le noyau fonctionnel. La mise en œuvre de ces interfaces multimodales en entrée regroupe la conception, le développement et l'exécution. Le dynamisme et l'hétérogénéité des environnements pervasifs naissants sont au premier plan. Le domaine d'application est l'informatique personnelle, notamment au sein du logis. Sans pour autant se limiter strictement à ce domaine, la généralisation aux lieux publics et aux environnements professionnels nécessitera des expérimentations dépassant le cadre de cette thèse.

Le cadre de ce travail implique nécessairement une approche pluridisciplinaire – deux directeurs de thèse, deux équipes, deux domaines : interaction homme-machine et génie logiciel – afin de confronter deux visions et leurs approches associées. De plus, au-delà d'une simple confrontation, l'objectif est d'identifier les points d'accroche et pour un domaine, quels sont les idées, approches et paradigmes utiles pour l'autre domaine ?

Vis-à-vis de ces objectifs de haut niveau d'abstraction, nos principales contributions sont à la fois conceptuelles et pratiques sous la forme d'un environnement logiciel.

En effet une contribution principale prend tout d'abord la forme de propositions conceptuelles : une caractérisation des dispositifs d'interaction en entrée, un langage pour la spécification d'interaction multimodale en entrée et un modèle à composant pour la multimodalité. Une architecture générale est aussi proposée : elle organise nos propositions dans un canevas, cohérent et pertinent vis-à-vis de l'objectif. Ces propositions conceptuelles reposent sur l'articulation de plusieurs approches en génie logiciel : service et médiation de données, informatique autonomique, ingénierie dirigée par les modèles, DSL [13], et informatique distribuée.

Ces propositions conceptuelles sont complétées par leur mise en œuvre concrète. Ainsi, une deuxième contribution de nos travaux prend la forme de réalisations logicielles : l'implémentation des propositions conceptuelles dans un ensemble d'outils logiciels. Ces outils sont stables

11. Spot publicitaire pour l'iPhone : `http://www.youtube.com/watch?v=pUfO3dJpcPA`.
12. Durant l'année fiscale 2009, le montant des dépenses publicitaires s'élèvent environ à 1,4 milliard de dollars pour Microsoft, 800 millions pour Dell et 500 millions pour Apple. Voir l'article *Apple's 2009 ad budget: Half a billion* par Philip Elmer-DeWitt, le 28 octobre 2009. `http://tech.fortune.cnn.com/2009/10/28/apples-2009-ad-budget-half-a-billion/`.
13. Sigle de *domain-specific language*, qui peut se traduire par « langage dédié ».

et utilisables et offrent une bonne couverture fonctionnelle: plus complets qu'un prototype de recherche mais sans avoir la prétention d'être une réalisation fournissant une fiabilité durement éprouvée, des performances optimales et une documentation complète.

Par ses contributions, ce travail doctoral définit une **approche globale** pour permettre des **interactions multimodales en entrée** dans les environnements où les dispositifs d'interaction et les applications sont **dynamiques** et **hétérogènes**.

1.3 Organisation du document

Comme le montre la figure 1.3, le document est organisé en deux parties. La première partie (chapitres 2 et 3) définit le cadre général de l'étude ainsi que nos motivations. Il s'agit d'une analyse des travaux existants en ingénierie de l'interaction multimodale et de l'informatique pervasive. La deuxième partie du manuscrit (chapitres 4, 5 et 6) est consacrée à nos contributions en ingénierie des environnements pervasifs multimodaux en entrée.

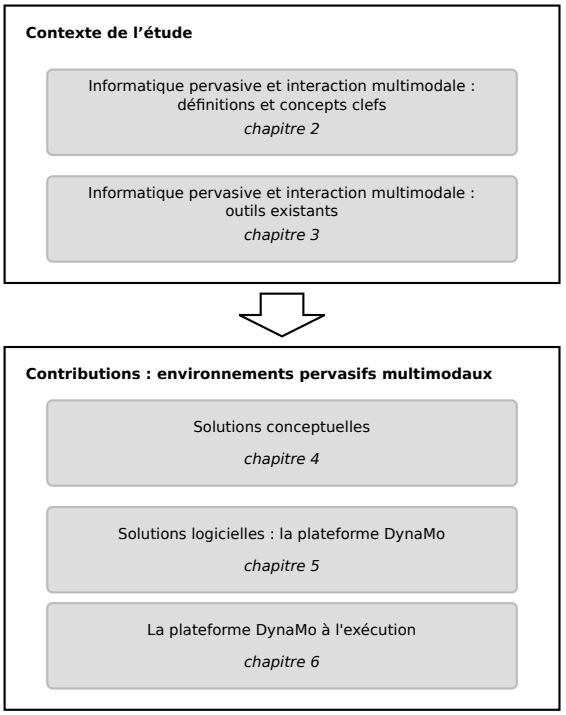


FIGURE 1.3 – Structure du document.

1.3.1 Présentation du contexte

Le chapitre 2 expose les définitions et concepts clefs de l'informatique pervasive et de l'interaction multimodale. Nous motivons ensuite nos travaux en explicitant les apports centraux de l'interaction multimodale aux environnements pervasifs. Nous concluons enfin en expliquant notre démarche pragmatique d'ingénierie pour le développement d'environnements pervasifs mul-

timodaux.

Nous démontrons dans le chapitre 3 que les outils actuels ne sont pas adaptés pour traiter les aspects centraux de l'interaction multimodale en environnements pervasifs. À des fins analytiques et selon une grille d'analyse commune établie à partir des concepts clefs du chapitre 2, nous présentons successivement les outils existants pour l'interaction multimodale, pour le dynamisme et pour l'informatique pervasive en général. Nous concluons ce chapitre en soulignant que la classification proposée est purement analytique et que les frontières ne sont pas aussi franches. Nous montrons néanmoins qu'aucun des outils existants ne répond aux besoins énoncés de l'interaction multimodale en environnements pervasifs.

1.3.2 Environnements pervasifs multimodaux : nos solutions conceptuelles et logicielles

Dans le chapitre 4, nous présentons nos solutions conceptuelles pour la conception et l'exécution d'interfaces multimodales en entrée qui prend notamment en compte deux caractéristiques primordiales des environnements pervasifs : l'hétérogénéité et le dynamisme. Notre contribution conceptuelle s'articule selon trois facettes complémentaires : un langage de spécification de l'interaction multimodale, un gestionnaire autonomique et une plateforme d'intégration. Notre proposition conceptuelle couvre à la fois la spécification et l'exécution d'une interface multimodale en environnements pervasifs.

Ces résultats conceptuels mettant en exergue la gestion autonomique de l'interaction multimodale sont publiés dans :

1. P-A Avouac, P. Lalanda and L. Nigay. Towards autonomic multimodal interaction. Actes de la conférence MAASC 2011, the 1st Workshop on Middleware and Architectures for Autonomic and Sustainable Computing, Paris, France, May 12, 2011, ACM Press, pp. 25-29.

Le chapitre 5 présente la plateforme logicielle conçue et réalisée : DynaMo. DynaMo implémente la solution conceptuelle énoncée au chapitre 4 en partant d'outils existants pour la gestion du dynamisme : iPOJO pour le modèle à composant et Cilia comme plateforme de médiation. Les technologies existantes utilisées ont toutes OSGi en commun. DynaMo hérite donc du dynamisme mais aussi de la robustesse fournie par OSGi.

Ces travaux sont publiés dans les deux communautés : interaction homme-machine (et en particulier l'interaction multimodale) et génie logiciel :

2. P-A Avouac, P. Lalanda and L. Nigay. Service-Oriented Autonomic Multimodal Interaction in a Pervasive Environment. Actes de la conférence ICMI'11, the 13th international conference on Multimodal interfaces, Spain, November 14-18, 2011, ACM Press, pp. 369-376.
3. P-A Avouac, P. Lalanda and L. Nigay. Adaptable multimodal interfaces in pervasive environments. Actes de la conférence CCNC 2012, IEEE Consumer Communications and Networking Conference, USA, 14-17 January, 2012, IEEE, pp. 544-548.

Tandis que le chapitre 5 est consacré à la conception et réalisation logicielle de la plateforme DynaMo donc à une vue statique de la plateforme DynaMo, nous étudions les aspects liés à l'exécution dans le chapitre 6. Ce chapitre traite deux volets complémentaires de l'exécution, qui soulignent à nouveau la pluridisciplinarité de nos travaux :

- le premier concerne les performances techniques de la plateforme, aspects classiquement traités en génie logiciel ;

– le deuxième volet présente un ensemble de cas d'interaction multimodale dynamique gérés par la plateforme, soulignant l'étendue des possibilités traitées par DynaMo d'un point de vue interaction homme-machine.

Ces travaux sont publiés dans :

4. P-A Avouac, P. Lalanda and L. Nigay. Autonomic Management of Multimodal Interaction: DynaMo in action. Actes de la conférence EICS 2012, the fourth ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Denmark, June 25-28, 2012, ACM Press, pp. 35-44.

Finalement, le chapitre 7 conclut le document. Ce chapitre résume les contributions principales de la thèse et reprend les résultats décrits dans les chapitres 4, 5 et 6. Des propositions de travaux futurs à court et à long terme sont également présentées.

Chapitre 2

Informatique pervasive et multimodalité

Nos travaux de recherche sont consacrés à l'interaction multimodale en environnement pervasif. Dans ce chapitre, l'informatique pervasive est définie, ainsi que les concepts de base de l'interaction multimodale. Puis, les apports et l'adéquation de la multimodalité pour les environnements pervasifs sont développés.

2.1 Informatique pervasive

En 1991, Mark Weiser a formulé la vision de l'informatique ubiquitaire dans l'article « The Computer for the 21st Century » [Weiser, 1991]. L'idée de l'informatique ubiquitaire est la disparition de l'informatique de la conscience des utilisateurs. Cette vision propose un monde où l'utilisateur n'est plus capable de discerner clairement quel matériel traite l'information. Les matériels informatiques sont de tailles diverses et sont disséminés dans l'environnement. Ils communiquent entre eux, ont une capacité de traitement de l'information, et sont capables de s'organiser pour répondre aux attentes de l'utilisateur. Les termes « informatique pervasive », « informatique ambiante » et « intelligence ambiante » désignent la même vision [Ark et Selker, 1999].

Quels sont les pré-requis dans le domaine de l'informatique pour la réalisation de cette vision ? En premier lieu, l'informatique se base sur des matériels physiques. Ainsi, l'électronique apporte des matériels ayant la capacité physique de traiter de l'information, de quantifier son environnement et d'agir dessus, et enfin de communiquer avec d'autres matériels. Si l'on se réfère à l'architecture classique de Von Neumann, la capacité de traitement de l'information est fournie par un processeur, une mémoire et une interface d'entrée et de sortie. Sur cette interface se branchent des transducteurs : des capteurs et des effecteurs. Ces transducteurs sont très divers : thermomètre, système de chauffage ou de ventilation, microphone, haut-parleur, photodétecteur, lampe électrique, moteur de volet, accéléromètre, récepteur GPS, antenne radioélectrique, etc. C'est à travers ses transducteurs que le matériel quantifie, agit et communique. Sur cette base physique, le système logiciel a pour objectif de répondre aux attentes de l'utilisateur.

Pour atteindre cet objectif, des tâches complexes sont nécessaires. Dans l'ordre chronologique :

- Identifier l'attente de l'utilisateur. L'utilisateur peut donner un ordre explicite au système, le système est alors dit passif. Mais le système peut aussi est proactif, c'est-à-dire déduire par lui-même l'attente de l'utilisateur.

– Inférer la suite d'actions à réaliser pour répondre à l'attente de l'utilisateur. Cette suite d'actions dépend en général du contexte d'exécution, notamment de l'utilisateur.
– Effectuer les actions.
– Éventuellement analyser la réponse donnée et inférer une manière plus adéquate de répondre à l'attente de l'utilisateur.

Outre la difficulté de répondre aux attentes des utilisateurs, l'informatique pervasive implique la plupart du temps un système logiciel dynamique et hétérogène. Il y a plusieurs raisons à cela : un utilisateur est mobile, il amène et emporte des matériels, un matériel peut être éteint, un matériel autonome peut se retrouver à court d'énergie, etc. Le système logiciel est ainsi un assemblage opportuniste, momentané, qui est réalisé lors de son exécution : il n'est pas connu dans son ensemble lors de sa conception. De plus, les matériels et les logiciels informatiques actuels sont hétérogènes, notamment au niveau des protocoles de communication. Ainsi, le système logiciel est hétérogène car il est composé d'un ensemble de logiciels hétérogènes.

Il en ressort plus généralement que le système logiciel doit s'adapter au contexte [Schilit *et al.*, 1994, Dey, 2001, Coutaz *et al.*, 2005], qui est hétérogène, dynamique, et non prévisible.

Auparavant, et encore maintenant dans une moindre mesure, l'informatique était très peu mobile, du fait de la taille et du poids des unités de traitement et des dispositifs d'entrée et de sortie. Les progrès en miniaturisation des composants électriques et électroniques, ainsi que l'essor des communications informatiques sans fils permettent une plus grande mobilité des matériels informatiques. Aussi, la baisse continuelle des prix des matériels informatiques et électroniques permettent à certains environnements personnels, publics et professionnels d'être peu à peu enrichis par divers systèmes informatiques. Ainsi, l'informatique semble devenir petit à petit pervasive. Par ailleurs, une partie de la communauté de la recherche en informatique donne de l'écho à la vision de l'informatique pervasive. En témoigne le nombre de références de l'article séminal [1], la création de conférences et revues internationales de haut niveau [2] relatives à ce sujet : les conférences UbiComp [3] en 1999, PerCom [4] et Pervasive [5] en 2003 ; le journal IEEE Pervasive Computing en 2002.

Les matériels existent, leur prix devient peu à peu abordable pour une partie non négligeable du monde. De nombreux systèmes logiciels sont créés pour tenter d'organiser ces matériels avec l'objectif de répondre aux besoins de l'utilisateur. Une partie de la communauté des chercheurs s'intéresse à la problématique. Pourtant, l'informatique pervasive reste une vision. En 2001, dans son article « Pervasive Computing: Vision and Challenges » [Satyanarayanan, 2001], Mahadev Satyanarayanan analysait les avancées dans les domaines afférents à l'informatique pervasive, ainsi que les problèmes restant à résoudre. Force est de constater qu'une grande partie de cet article aurait pu être écrit aujourd'hui. La raisons évoquée par l'auteur de la difficulté de réaliser la vision de l'informatique pervasive est que « le tout est supérieur à la somme de l'ensemble des parties » [6]. C'est-à-dire que les efforts doivent être effectués dans l'intégration des technologies hétérogènes. Ainsi, la difficulté réside dans l'architecture, la combinaison des composants et l'ingénierie au niveau du système. L'article est centré sur les capacités des systèmes informatiques, l'auteur laisse volontairement de côté les problèmes liés à l'interaction entre l'humain et les

1. 9200 papiers scientifiques référencent l'article, d'après Google Scholar en novembre 2012.
2. Conférences de rang A selon le classement ERA de 2010, voir `http://www.arc.gov.au/era/era_2010/archive/era_journal_list.htm`.
3. ACM International Conference on Ubiquitous Computing, appelée « Handheld and Ubiquitous Computing » les deux premières années.
4. IEEE International Conference on Pervasive Computing and Communications.
5. International Conference on Pervasive Computing.
6. Traduction de l'anglais « the whole is much greater than the sum of its parts ».

machines, pour se concentrer sur l'aspect génie logiciel.

La communauté des chercheurs du domaine de l'interaction homme-machine (IHM) propose plusieurs approches, que nous détaillons dans les paragraphes suivants, pouvant être appliquées à l'informatique pervasive. Ces approches n'ont pas nécessairement été proposées pour résoudre les problèmes de l'informatique pervasive : certaines approches avaient été proposées avant 1991. Elles peuvent être complémentaires. Pour faire face à un environnement inconnu lors de la spécification, la construction automatique d'interfaces graphiques à l'exécution est proposée. Cette construction se fait en fonction des services auxquels souhaite accéder un utilisateur [Ponnekanti *et al.*, 2001, Gabillon *et al.*, 2011]. La distribution et la migration d'interfaces permettent de tenir compte du contexte dynamique d'utilisation [Balme *et al.*, 2004]. La distribution d'une interface est l'utilisation de plusieurs ressources d'interaction par une interface. Par exemple, une interface graphique peut être découpée en plusieurs parties, chaque partie étant affichée sur des ressources distinctes. La migration est le transfert d'une interface présente sur une ressource vers une autre ressource. La conception et l'analyse de nouvelles techniques d'interaction [Scoditti *et al.*, 2011] permettent d'améliorer l'exploitation de nouveaux matériels manipulables par les humains. L'utilisation multiple de dispositifs d'interaction – la multimodalité – est une manière d'exploiter la profusion naissante de dispositifs d'interaction.

Les matériels informatiques peuvent être classés de différentes manières. Du point de vue de l'IHM, il fait sens de distinguer les matériels qui constituent une interface entre le système et l'utilisateur. De tels matériels sont appelés dispositifs d'interaction. Un dispositif d'interaction d'entrée est un matériel qui possède au moins un capteur sur lequel un utilisateur est capable d'agir directement pour transmettre une information au système informatique. « Directement » s'entend ici comme la modification par l'utilisateur d'au moins une grandeur physique captée par le dispositif. Un dispositif d'interaction de sortie est un matériel qui possède au moins un effecteur avec lequel un système informatique est capable de transmettre directement une information à un utilisateur. « Directement » s'entend ici comme la modification par le dispositif d'au moins une grandeur physique qu'un utilisateur peut percevoir via un ou plusieurs sens perceptifs humains. Le clavier, la souris, l'écran et les haut-parleurs sont les dispositifs d'interaction très répandus attachés aux ordinateurs personnels des années 1990. Le *trackball*, le pavé et la tablette tactile, le *touchpad* et le *trackpoint* ont été présents dans une moindre mesure. De nombreux dispositifs d'interaction ont été imaginés et prototypés depuis la naissance de l'IHM : le crayon optique *SketchPad* de Sutherland en 1964 [Sutherland, 1964], le casque à affichage en trois dimensions en 1968 [Sutherland, 1968], les systèmes de reconnaissance de parole [Juang et Rabiner, 2004] et de gestes [Mitra et Acharya, 2007], les techniques de suivi du regard [Duchowski, 2002, Morimoto et Mimica, 2005], les interactions entre les machines et le cerveau [Moore, 2003], etc.

L'informatique pervasive pose de nombreuses questions : comment un utilisateur peut-il connaitre les capacités d'un système informatique puisque celui-ci n'est plus clairement identifiable ? Comment l'utilisateur explicite-t-il son but s'il ne sait pas quel matériel est capable de lui procurer un service ? Du point de vue opposé, comment un système informatique peut-il être capable de déduire l'attente d'un utilisateur ? Le système informatique évolue, son contexte évolue, quels mécanismes permettent au système informatique d'utiliser au mieux ses propres capacités ? Chacune de ces questions en amène d'autres. Néanmoins, deux propriétés de l'informatique pervasive transparaissent dans chaque question : l'hétérogénéité et le dynamisme. Cette section se poursuit par une mise en lumière de ces deux propriétés des environnements pervasifs.

2.1.1 Hétérogénéité

Le nombre de matériels informatiques au sein d'un même environnement tend à augmenter. Ces matériels peuvent être prévus pour fonctionner ensemble, mais à moins d'en faire un critère de choix obligatoire lors de l'acquisition, et donc réduire drastiquement l'éventail de choix possibles, ils ne fonctionnent pas tous ensemble : ils ne sont pas interopérables. Concernant la communication, on peut distinguer deux contraintes d'interopérabilité. D'une part, l'information doit pouvoir circuler entre deux matériels, d'autre part, l'information doit être comprise par les deux entités communicantes.

2.1.1.1 Protocoles de communication

L'informatique est le traitement automatisé de l'information. L'information reçue est modifiée et transmise. Au sein d'un logiciel écrit avec un seul langage de programmation, la transmission de l'information est spécifiée par le programmeur en utilisant le formalisme du langage. Le compilateur ou l'interpréteur est en charge de mettre en place la transmission de l'information, de telle sorte que la sémantique du langage soit respectée. D'une instruction à l'autre, une information peut être transférée dans un registre du processeur. Un appel de fonction peut déclencher la mise sur une pile d'un paramètre. La sémantique donnée à un langage de programmation certifie au programmeur le bon fonctionnement du transfert des informations. La mise en œuvre de la communication est plus complexe lorsqu'il s'agit de transfert entre deux logiciels. Pour assurer la transmission et l'interprétation correcte des informations entre deux entités, un ensemble de règles doit alors être employé, c'est-à-dire un **protocole de communication**.

Le terme « protocole de communication » recouvre des réalités très différentes. Par exemple le protocole de communication RS-232 – souvent appelé liaison série – définit le type de broche, le nombre de fils électriques physiques et la vitesse de communication en fonction de la longueur des fils, le codage des bits sous forme de tension électrique : hauteur et durée du signal pour chaque bit, le découpage du flot de bits en trame, etc. Le protocole Ethernet définit le type de nommage des différentes entités par les adresses MAC [7], le découpage du flux de bits en trame, la manière de calculer le code d'intégrité pour une trame et son emplacement au sein de la trame, etc. Le protocole Bluetooth est un regroupement de plusieurs protocoles qui définissent des règles de très bas niveau comme la bande de fréquences des ondes électromagnétiques, des règles de plus haut niveau comme l'organisation logique d'un réseau en maitres et esclaves, jusqu'à des règles de haut niveau définissant des profils de matériels, la manière de découvrir les dispositifs accessibles, etc. Le protocole D-Bus est dédié à la communication entre processus, proposant un annuaire centralisé des services accessibles. Les règles définissent la manière d'interroger l'annuaire, la manière d'appeler un service, la manière de transmettre des informations à un service, etc.

L'impossibilité de communication due à l'emploi de protocoles différents peut être surmontée de manière logicielle. La seule exception est pour les protocoles spécifiant le plus bas niveau de communication : si les deux entités ne communiquent pas sur le même médium, l'énergie émise par l'une ne peut être récupérée par l'autre. Par exemple, une émission dans les câbles du réseau électrique ne peut pas être captée par un récepteur infrarouge, ou bien une antenne ne peut capter une onde électromagnétique qui n'est pas dans la plage des fréquences attendues. Cette barrière de communication est inhérente à la physique, et ne peut être surmontée par une solution logicielle. Dans ces cas là, il est nécessaire de disposer d'au moins un matériel capable de relayer l'information en faisant office de passerelle. Par exemple, pour récupérer des informations de la Wiimote qui utilise le protocole Bluetooth, il faut nécessairement avoir un récepteur physique Bluetooth.

7. Acronyme de *media access control*.

Ces impossibilités de communication ne sont pas un problème à petite échelle. La Wiimote a été conçue comme une manette de jeu, et elle est bien évidemment compatible avec la console qu'elle accompagne. La télécommande prévue pour une télévision est compatible avec cette même télévision. Cependant, à l'échelle d'un environnement, une pièce par exemple, les principes de l'informatique pervasive ne sont pas mis en oeuvre : seuls quelques « ilots » de communication apparaissent. La transparence, c'est-à-dire le fait de ne pas se soucier du dispositif à utiliser, n'est pas atteinte pour l'utilisateur : tel dispositif ne peut contrôler que tel matériel, et inversement, tel matériel ne peut être contrôlé que via tel dispositif. Un travail supplémentaire est nécessaire pour résoudre ces impossibilités de communication. Parfois, telle application installée sur tel téléphone qui a tel système d'exploitation permet de contrôler telle application. C'est typiquement le cas des logiciels de contrôle de logiciel de présentation pour les téléphones portables. Si ce type de logiciel a le mérite d'apporter une solution pour des cas particuliers d'incompatibilité, il n'apporte pas de solution à l'échelle de l'environnement : ce n'est pas une approche globale.

Si un utilisateur veut se construire un environnement pervasif, il doit acheter des matériels spécifiques. En achetant uniquement des matériels qui respectent telle norme, voir uniquement des matériels vendus par tel industriel dont les protocoles de communication ne sont pas ouverts. L'utilisateur est en un sens captif de la technologie, car ses acquisitions sont guidées par des compatibilités technologiques. D'autre part, à l'échelle de plusieurs environnements pervasifs, comme certains matériels sont transportables, un matériel qui a fonctionnement optimal dans un environnement homogène se retrouve incapable de se fondre dans un autre environnement. Le cas typique est la diffusion de musique : arrivé chez un ami, l'utilisateur ne peut diffuser sa musique à cause d'incompatibilités technologiques.

2.1.1.2 Dispositifs d'interaction

L'hétérogénéité concerne également les dispositifs d'interaction. Auparavant, les dispositifs d'entrée étaient très peu variés : principalement le clavier et la souris. D'autres dispositifs pouvaient être facilement utilisés en simulant soit la souris (trackball, tablette tactile, touchpad, trackpoint), soit le clavier (tablette tactile couplée avec un système de reconnaissance de caractères). Les concepteurs de ces matériels avaient en charge de créer un pilote capable d'être vu comme une souris ou un clavier par le système d'exploitation. Dès lors qu'une application prenait en compte un clavier ou une souris, aucun travail supplémentaire n'était nécessaire pour le développeur d'interface utilisateur pour prendre en compte les autres dispositifs. La souris et le clavier étant des standards de fait, le système d'exploitation et les boites à outils [8] pour le développement d'interfaces utilisateur incluent la récupération des évènements de bas niveau, pour les présenter à un niveau d'abstraction plus pratique pour le concepteur de systèmes informatiques. L'arrivée de nouveaux dispositifs non réductibles à un clavier ou une souris change le panorama et ces hypothèses. Cette nouvelle hétérogénéité nécessite de nouveaux outils, méthodes et architectures pour être prise en compte [Lorenz *et al.*, 2008]. Sans ce travail, la prise en compte de tous les dispositifs doit être faite pour chaque application. Il semble alors très probable que la transparence des dispositifs d'interaction promise par l'informatique pervasive ne puisse être réalisée. Le manque d'approche globale se répercute négativement sur l'utilisateur, sur le concepteur d'applications et sur le concepteur de dispositifs. L'utilisateur ne peut utiliser ses dispositifs avec n'importe quelle application, mais avec seulement les applications qui prennent en compte le dispositif. Ainsi, l'incitation à acheter un dispositif d'interaction reste faible. Le concepteur d'applications et le concepteur de dispositifs font face au même problème vu de chaque extrémité. En effet, d'un côté, l'application doit pouvoir découvrir à l'exécution

8. Une boite à outils graphique est une bibliothèque qui fournit un ensemble d'éléments graphiques et leurs comportements associés : boutons, zones de texte, conteneurs, etc.

un grand nombre de dispositifs, interpréter les données qui en sont issues, et gérer la disparition des dispositifs. De l'autre côté, le dispositif doit pouvoir être reconnaissable et utilisable par un grand nombre d'applications.

Pour que l'ensemble hétérogène de ces dispositifs d'interaction puissent être pris en compte, il est nécessaire d'avoir un moyen d'exprimer des équivalences du point de vue des applications entre ces dispositifs ou entre des parties de dispositifs. Ces équivalences peuvent être soit explicites, soit déduites d'une description de dispositif. Dans la suite de cette section, sont présentés les approches de modélisations des dispositifs, puis les standards industriels les plus couramment utilisés.

2.1.1.2.1 Travaux de recherche en modélisation des dispositifs d'interaction

En 1968, Newman propose un formalisme séminal qui permet de décrire une interaction graphique sous forme d'automates à états finis [Newman, 1968]. Un état correspond à un état du système interactif, une transition entre deux états représente une action. Une action est initiée par l'utilisateur ou par le système. Une action fait partie d'une catégorie. Une catégorie regroupe des actions qui sont **compatibles**. Un exemple de catégorie est la réalisation par l'utilisateur d'une action « mouvement de crayon ». Cela peut être réalisé par un mouvement de crayon optique ou de trackball, ou par deux nombres saisis sur un clavier, ou encore par deux nombres lus sur un support d'enregistrement. D'autres catégories sont spécifiées : la catégorie « commande » qui peut être réalisée par l'appui d'un bouton graphique ou par une suite de caractères saisie au clavier, l'utilisation de cette catégorie nécessite un paramètre qui correspond au nom de la commande ; la catégorie « bouton » qui correspond à l'appui sur un bouton physique ; la catégorie « pointage de crayon » qui correspond à l'appui d'un crayon optique sur un point de l'écran ; et la catégorie « système » qui correspond aux actions provenant du système. De nouvelles catégories peuvent être créées par un concepteur. Ainsi, les catégories constituent un moyen d'établir une indépendance entre la modélisation de l'interaction et les dispositifs d'interaction. À chaque entrée reçue par le système interactif, un gestionnaire de réaction est chargé de la faire correspondre à une action, et ainsi trouver sa catégorie. Le formalisme de description des actions apporte deux fonctionnalités notables. D'une part, il permet de concaténer les entrées pour n'en faire qu'une seule action, par exemple pour la catégorie « commande », il est nécessaire de grouper les caractères pour obtenir le nom complet de la commande. D'autre part, il permet d'associer une catégorie à une action, c'est-à-dire expliciter des équivalences entre actions. Par exemple pour la catégorie « commande », la frappe du nom de la commande est équivalente à l'appui d'un bouton graphique lié à cette commande. Les données d'un même capteur peuvent être interprétées comme des actions différentes, donc comme appartenant à des catégories différentes : dans l'exemple donné, le crayon optique permet de faire des actions faisant partie des catégories « pointage de crayon » et « commande ». Le découplage de la description des catégories et de la description du système interactif permet de réutiliser les catégories et les actions dans un autre programme, qui a une description propre de son système interactif. Ce travail de Newman avait pour objectif premier de faciliter le développement de systèmes interactifs ; l'indépendance et l'équivalence des dispositifs sont des propriétés de la solution proposée.

En 1976, Wallace traite explicitement des problèmes posés aux développeurs d'applications liés à l'hétérogénéité des dispositifs d'interaction [Wallace, 1976]. Il propose un ensemble réduit de dispositifs virtuels d'entrée qui permet de modéliser dans les programmes tous les dispositifs d'entrée. À l'image des catégories proposées par Newman, les dispositifs virtuels doivent facilement **remplacer** des dispositifs physiques connus ; et les possibilités de tous les dispositifs physiques doivent pouvoir **être représentées** par au moins un dispositif virtuel. Cet ensemble

de dispositifs virtuels permet de définir la sémantique des entrées indépendamment de la forme physique des dispositifs. Un premier ensemble de dispositifs, qualifiés de primitifs, en comporte cinq : le choix, le bouton, le clavier, le localisateur et le valuateur [9]. Le choix permet de désigner un objet préalablement défini par un utilisateur, son prototype physique est le crayon optique. Le bouton est utilisé pour sélectionner un objet préalablement défini par un programme, son prototype physique est le bouton programmé sur un clavier. Le clavier permet de saisir du texte, son prototype physique est le clavier informatique. Le localisateur permet d'indiquer une position ou une orientation dans l'espace de l'écran d'affichage, ses prototypes physiques sont la souris, la tablette et le joystick. Le valuateur permet de déterminer une valeur au sein d'un espace vectoriel des nombres réels, son prototype physique à une dimension est le potentiomètre. Ces dispositifs primitifs sont caractérisés par : la sortie de l'échantillonnage, les évènements possiblement émis ainsi que les informations qui y sont attachées, le retour visuel et une liste de paramètres de configuration qui permettent au programme de modifier le comportement du dispositif [10]. Par exemple, un valuateur échantillonné produit un vecteur réel, les évènements liés sont soit une sélection qui fournit la valeur courante, soit un dépassement qui fournit une valeur et la borne dépassée, le retour visuel est l'affichage de la valeur courante, sa configuration contient la dimension, c'est-à-dire le nombre de valeurs fournies en même temps, l'activation du retour visuel et l'intervalle des valeurs acceptées.

En 1983, Buxton reconnait le bénéfice pour le programmeur de développer une interaction en fonction de dispositifs virtuels, mais reproche à cette solution de ne pas assez prendre en compte les particularités de chaque dispositif [Buxton, 1983]. Il propose donc une taxonomie qui apporte des informations supplémentaires, de sorte à pouvoir faire des équivalences entre dispositifs plus appropriés. La taxonomie ne modélise que les dispositifs d'interaction contrôlés avec les mains, qui sont continus et non discrets. Cette taxonomie peut être représentée par un tableau à deux dimensions, les dispositifs sont placés dans les cellules selon leurs particularités. Une colonne représente le nombre de dimensions captées par le dispositif, une ligne représente la propriété physique captée. Le nombre de dimensions va de une à trois. Les propriétés captées sont soit la position, soit le mouvement, soit la pression. La position et le mouvement sont raffinés en mécanique et tactile. Les colonnes sont divisées informellement par le type de geste que doit faire l'utilisateur. Par exemple, la souris est un dispositif d'interaction qui capte le mouvement, elle produit des valeurs à deux dimensions, son mouvement est mécanique, et son utilisation est plus proche de celle d'une tablette que de celle du crayon optique. Le tableau 2.1 situe quelques dispositifs dans la taxonomie.

		Number of Dimensions						
		1		2		3		
Property Sensed	Position	Rotary Pot	Sliding Pot	Tablet	Light Pen	Joystick	3D Joystick	Mechanical
				Touch Tablet	Touch Screen			Touch Sensitive
	Motion	Continuous Rotary Pot	Treadmill Thumbwheel	Mouse		Trackball	Trackball	Mechanical
			TASA Ferinstat			TASA X-Y Pad		Touch Sensitive
	Pressure	Torque Sensing	Pressure Pad			Isometric Joystick		

Table 2.1 – Taxonomie de Buxton. Adapté de [Buxton, 1983].

9. Proposition de traduction pour : *pick*, *button*, *keyboard*, *locator* et *valuator*.
10. Proposition de traduction pour : *sampling output*, *event*, *event report*, *echo*, *attribute*.

En 1984, Foley, Wallace et Chan proposent une classification des dispositifs et techniques d'interaction d'entrée dans le but d'aider les concepteurs à choisir ou combiner les techniques d'interaction [Foley *et al.*, 1984]. Ils se focalisent sur les interactions graphiques pour la clarté et la richesse de communication qu'elles permettent. Pour cela, ils isolent six types de tâches d'interaction, qui sont indépendantes des applications et des dispositifs physiques : sélection, positionnement, orientation, saisie de chemin, quantification et saisie de texte [11]. Ces types de tâches sont basés sur l'expérience des auteurs vis-à-vis des systèmes d'interaction graphique. Chaque type de tâches peut être réalisé par différentes techniques, chaque technique peut être à son tour réalisée au moyen de différents dispositifs d'interaction. Chaque type de tâches a également un ensemble de prérequis pour l'application. Une tâche de sélection permet de sélectionner un élément parmi un ensemble de choix possibles. Une tâche de positionnement permet d'indiquer une position sur une surface d'affichage. Une tâche d'orientation permet d'orienter une entité dans un espace en deux ou trois dimensions. Une tâche de saisie de chemin permet de générer une série de positions ou d'orientations. Une tâche de quantification permet de spécifier une valeur numérique. Une tâche de saisie de texte permet évidemment de saisir du texte. Une tâche de quantification peut être réalisée par une technique directe avec un dispositif « quantifieur », au moyen d'un potentiomètre linéaire ou rotatif. Elle peut aussi être réalisée par une technique de saisie de chaines de caractères, au moyen d'un clavier ou de la parole. D'autres techniques peuvent être utilisées pour réaliser ce type de tâches. Deux prérequis doivent être spécifiés par une application pour une tâche de quantification : la résolution et la fermeture ou la répétition de la plage de données. La résolution est le nombre de valeurs différentes possibles.

En 1990, Mackinlay, Card et Robertson proposent une manière de décrire les dispositifs d'interaction d'entrée [Mackinlay *et al.*, 1990, Card *et al.*, 1991]. Le but est de pouvoir analyser l'espace de conception des dispositifs d'interaction [12]. Un dispositif est modélisé par six caractéristiques, ou par une composition de dispositifs. Trois opérateurs de composition sont proposés. Les six caractéristiques d'un dispositif concernent :

- Le type de la propriété physique mesurée : position ou force, absolue ou relative, linéaire ou rotative. Par exemple, un bouton rotatif de réglage de volume sonore mesure une position absolue rotative.
- La fonction de correspondence entre les valeurs de la propriété physique mesurée et les valeurs de sortie du dispositif. Par exemple, un bouton rotatif qui permet de sélectionner un choix parmi trois et qui a une course entre 0 et 90 degrés peut avoir pour fonction de correspondance : $e \in [0, 22.5] \Rightarrow s = 0 \quad \vee \quad e \in]22.5, 67.5[\Rightarrow s = 45 \quad \vee \quad e \in [67.5, 90] \Rightarrow s = 90$, où e est la valeur d'entrée et s la valeur de sortie.
- L'ensemble de départ et l'ensemble d'arrivée de la fonction de correspondance. Dans l'exemple précédent, l'ensemble de départ est [0, 90], l'ensemble d'arrivée est {0, 45, 90}.
- La caractérisation de son état courant.
- Un ensemble de propriétés qui permettent de décrire le fonctionnement du dispositif qui ne peut être modélisé par la fonction de correspondance. Par exemple si le bouton rotatif dispose de crans qui donne l'impression à l'utilisateur de manipuler un dispositif discret, et que l'état du dispositif contient un booléen reflétant le fait que le bouton est en mouvement, une propriété peut spécifier que la sortie est inchangée si le bouton est en mouvement.

Les dispositifs de la même famille peuvent être modélisés par un dispositif générique. Une relation d'instanciation permet de spécialiser des dispositifs à partir d'un dispositif générique. Les opérateurs de composition permettent de modéliser :

- la connexion de dispositifs : les valeurs de sortie d'un dispositif deviennent les valeurs

11. Proposition de traduction pour : *select*, *position*, *orient*, *path*, *quantify*, *text*.

12. L'espace de conception peut être vu comme l'ensemble des dispositifs d'interaction pouvant être conçu. Cet espace existe indépendamment de la manière de le décrire.

d'entrée du dispositif connecté ;
- le regroupement de dispositifs : un clavier est un regroupement de tous ses boutons, la sortie du clavier est l'union des sorties de tous les boutons, c'est-à-dire que pour un clavier à n touches, la sortie est n valeurs dans un espace à une dimension ;
- la fusion de dispositifs : une tablette est une fusion de deux tablettes à une dimension, sa sortie est une valeur dans un espace à deux dimensions.

Le résultat d'une composition de deux dispositifs est un dispositif.

2.1.1.2.2 Standards industriels

L'USB [13] est un standard de communication entre un ordinateur et un dispositif électronique qui est apparu au milieu des années 1990 [usb20, 2000]. Il est maintenant très répandu : en 2009, plus de trois milliards de dispositifs USB sont sur le marché et, chaque année, deux milliards de dispositifs sont vendus [14]. Le protocole définit plusieurs classes de dispositifs, par exemple celle des dispositifs d'interaction, celle des dispositifs liés au son, celle des dispositifs liés à l'image, etc. Une classe définit la manière de transmettre les données entre l'hôte et le dispositif. Un seul pilote par classe est nécessaire. Ainsi, la classe des dispositifs d'interaction [usbhid111, 2001] recouvre un large éventail de dispositifs parmi lesquels : les claviers et dispositifs de pointage, les boutons et interrupteurs, les gants de données, les dispositifs prévus pour la simulation (pédales, volants...) et les jeux vidéos, les dispositifs de contrôle d'interfaces graphiques, de son, de vidéo, ainsi que les dispositifs qui fournissent des données similaires, comme les thermomètres et les lecteurs de code-barres. Cette classe de dispositifs a trois buts notables qui facilitent la gestion de l'hétérogénéité : la possibilité pour les applications de ne pas tenir compte d'informations inconnues, l'extensibilité et l'auto-description. Un seul pilote est nécessaire dans le système d'exploitation pour communiquer avec tous les dispositifs qui déclarent faire partie de cette classe. Le pilote est en charge de ventiler les données qu'il reçoit du dispositif vers la partie logicielle adéquate. Il utilise pour cela des informations fournies par le dispositif lui-même. Ces informations caractérisent le codage des données, la grandeur physique captée, les bornes de l'intervalle de valeur logique et physique à partir desquels il est possible de calculer la résolution, le fait que les données sont relatives ou absolues, le comportement des données lorsqu'elles arrivent à une borne de l'intervalle, etc. Les relations entre les données provenant de plusieurs capteurs peuvent être explicitées. Une relation peut refléter la proximité physique des capteurs (pour un accéléromètre qui produit les accélérations sur trois axes), la cohésion logique (les boutons d'une souris), etc. Pour permettre aux applications d'interpréter correctement les données reçues, un dispositif présente également la signification des données qu'il envoie [usbhid112, 2004]. Ces significations portent sur des données simples (comme la vitesse d'un club de golf, l'ordre d'éjecter un média d'un lecteur, l'ordre de mettre une communication téléphonique en attente, créer un nouveau document) et des ensembles de données (comme un téléphone, une souris, un gant de données).

Le standard Bluetooth [bluetooth4.0, 2010] est une spécification pour la communication sans fil entre des dispositifs électroniques. Chaque année, plus de deux milliards de circuits intégrés Bluetooth sont mis sur le marché [15]. Il a été créé au milieu des années 1990, la version 1.0 de la spécification est publiée en 1999. L'interopérabilité entre les dispositifs est réalisée grâce au concept de profil. Le standard Bluetooth définissant un ensemble de protocoles de communication, un profil définit quels protocoles sont utilisés, quel format une donnée doit avoir, et la

13. USB est l'acronyme de *universal serial bus*, qui peut se traduire par « bus série universel ».
14. Selon l'article « SuperSpeed USB 3.0: More Details Emerge » par Melissa J. Perenson, le 7 janvier 2009, sur PC World. `https://www.pcworld.com/article/156494/superspeed_usb_30_more_details_emerge.html`.
15. Selon la section *Smart home market* du site du Bluetooth SIG, `http://www.bluetooth.com/Pages/Smart-Home-Market.aspx`.

manière de faire la découverte de dispositifs. Un profil particulier recouvre de nombreux dispositifs d'interaction : le profil HID [16] [bluetoothhid10, 2003]. Ce profil HID reprend les concepts de la classe HID de l'USB. Ce profil repose donc sur l'extensibilité, l'auto-description par un dispositif des données qu'il envoie, et la possibilité pour les applications de ne pas tenir compte d'informations qu'elles ne peuvent interpréter.

2.1.2 Dynamisme

Un environnement pervasif évolue, plusieurs facteurs déclenchent ces évolutions. La configuration physique peut être modifiée : mise en place, déplacement ou retrait de matériels électroniques. L'environnement logiciel change également : installation, mise à jour ou désinstallation de logiciels, mais aussi accès à un nouveau service distant ou disparition du type de services. Ces évolutions sont aussi déclenchées par les actions d'un utilisateur : changement de la configuration d'un système, mise en marche ou arrêt d'un dispositif, changement de ses attentes, etc. Enfin, des facteurs externes peuvent modifier l'environnement pervasif, comme un dispositif à court d'énergie, un problème logiciel ou matériel qui rend un dispositif inutilisable, etc. Ces diverses évolutions possibles impliquent qu'un environnement pervasif est dynamique.

Lorsqu'un système est susceptible d'être dynamique, un compromis est à trouver par le concepteur entre la complexité de la mise en œuvre de la prise en compte de ce dynamisme d'une part, et le confort apporté à l'utilisateur final d'autre part. Par exemple, au niveau du système d'exploitation, un utilisateur peut vouloir passer rapidement d'une application à une autre : la mise en œuvre du multitâche est très complexe, mais le confort apporté à l'utilisateur est tel que la plupart des systèmes d'exploitation actuels sont multitâches. La vision de l'informatique pervasive met clairement l'accent sur l'organisation **automatique** des dispositifs électroniques. Ainsi, doit être prévalent le confort de l'utilisateur, qui n'a pas à gérer manuellement les changements au sein de l'environnement. De plus, l'opportunisme voulu pour les systèmes pervasifs suppose d'avoir une vision de l'état actuel de l'environnement, ce qui implique de suivre les évolutions.

La gestion du dynamisme au sein d'un système est en général difficile à réaliser. Un système est bien souvent spécifié à une ou plusieurs étapes de son cycle de vie. Logiquement, une spécification n'est utile que si elle est intangible. Or, spécifier le dynamisme, c'est expliciter les évolutions possibles au sein du système. Si une spécification peut énumérer les cas d'évolution possible, ces cas peuvent éventuellement se combiner s'ils ne sont pas indépendants, et déboucher sur une explosion combinatoire du nombre de cas à spécifier. C'est typiquement le cas des systèmes pervasifs : ne serait-ce qu'au niveau de l'assemblage de différents matériels, le nombre d'assemblages possibles est très grand.

Par ailleurs, l'importance de l'impact des évolutions est à considérer. Si les paramètres de configuration d'un logiciel ont généralement un impact faible sur le système, tel le choix de la langue d'un dictionnaire dans un éditeur de texte, d'autres peuvent avoir un impact très important : la disparition d'un dispositif peut amputer le système d'une fonctionnalité majeure qui ne peut être remplacée. De plus, les évolutions peuvent être très différentes les unes des autres, donc éventuellement avoir des impacts à différents niveaux d'un système. Par exemple, le paramétrage de la langue du dictionnaire n'a pas le même impact que le paramétrage de la langue de l'interface graphique, qui elle peut être gérée au niveau du système d'exploitation. Ensuite, gérer les évolutions peut être rendu encore plus compliqué si les évolutions possibles ne sont pas clairement connues à l'avance, ce qui est le cas des systèmes pervasifs.

16. HID est l'acronyme de *human interface device*, qui peut se traduire par « dispositif d'interface avec un humain ».

Beaucoup de logiciels proposent via leur interface de configuration de modifier leur fonctionnement. Ils définissent ainsi clairement les évolutions possibles, et sont directement avertis des changements. À l'opposé, un logiciel qui réagit aux évolutions de l'environnement n'est pas directement averti des modifications. Il doit lui-même s'enquérir de l'état de l'environnement et en déduire les évolutions. Ces applications sont dites sensibles au contexte [Schilit *et al.*, 1994]. Un problème de premier plan apparait alors : comment s'enquérir de l'état de l'environnement ? Deux possibilités existent : soit découvrir par lui-même les évolutions, soit s'inscrire à un mécanisme de notifications si celui-ci existe. Ces deux possibilités sont complémentaires, car l'hétérogénéité actuelle au sein des environnements ne permet pas de couvrir par un seul mécanisme toutes les évolutions. Par exemple, la découverte de dispositifs est bien souvent faite via un protocole de découverte. Néanmoins, l'hétérogénéité des protocoles de communication impose d'utiliser plusieurs protocoles de découverte pour couvrir un large champ de dispositifs [Zhu *et al.*, 2005]. La plupart du temps, le protocole de découverte est étroitement associé au protocole de communication. C'est le cas de Bluetooth avec le *service discovery protocol*[17], d'UPnP avec le *simple service discovery protocol*[18], etc. Un mécanisme de notifications peut être fourni par le système d'exploitation : Windows par exemple fourni un système de notification pour les applications, qui leur permet de suivre les évolutions des dispositifs connectés et déconnectés [19].

Certes, la présence de dispositifs peut être obtenue en exploitant divers mécanismes de découvertes complémentaires. Mais l'environnement ne recouvre pas seulement la présence des matériels informatiques, mais aussi les utilisateurs, l'emplacement géographique, etc. Schilit et al. par exemple, définissent le contexte comme l'endroit de l'interaction, les personnes alentour, les hôtes et les dispositifs accessibles, mais aussi les conditions lumineuses et sonores, la présence d'une connexion au réseau, son débit et son cout, et la situation sociale de l'interaction [Schilit *et al.*, 1994]. Ces informations sont de natures très diverses, et ne sont pas directement accessibles comme peut l'être la présence de dispositifs : les conditions sonores et lumineuses peuvent être récupérées via les capteurs idoines, la présence de personnes peut être déduite de multiples manières, plus ou moins directement selon le type de capteurs à disposition, la situation sociale de l'interaction peut être très difficile à appréhender de manière automatique. La difficulté réside ici non pas dans le suivi des évolutions, mais dans la sélection et l'utilisation des données issues des capteurs qui permettent d'inférer des informations de haut niveau d'abstraction.

Tous ces aspects du dynamisme des environnements pervasifs doivent potentiellement être pris en compte, et donc influer sur l'exécution d'un système. Les **adaptations** à réaliser sont plus ou moins complexes. Le degré de complexité peut être grossièrement assimilé à la proportion d'entités impactées nécessaire pour réaliser l'adaptation, et la manière de les effectuer (reconfiguration, changement d'architecture...). Un exemple très connu d'adaptation au changement de luminosité ambiant est le changement de la luminosité d'un écran. De même, si le niveau sonore ambiant dépasse un certain seuil, le niveau sonore de sortie d'une application peut être augmenté, ou de manière plus complexe l'information à transmettre à l'utilisateur peut devenir visuelle plutôt que sonore. L'apparition d'un nouveau dispositif d'interaction peut permettre de coupler ce dispositif avec un dispositif d'interaction déjà présent. Au-delà de l'adaptation de la forme des entrées et des sorties d'un système interactif, les adaptations possibles sont très diverses : gestion des priorités d'envoi de courriels, transfert de fichiers entre différentes machines [Garlan *et al.*, 2002], etc.

Différentes méthodes existent pour gérer le dynamisme. L'adaptation peut être statique ou

17. Voir *part B, Service Discovery Protocol (SDP) Specificaton* de la spécification Bluetooth [bluetooth4.0, 2010].

18. Voir *section 1, Discovery* de la spécification UPnP [upnp11, 2008].

19. Selon la section *About Device Management* de la bibliothèque MSDN, sur le site de Microsoft. Voir `http://msdn.microsoft.com/en-us/library/aa363142(v=VS.85).aspx`.

dynamique. L'adaptation statique consiste à gérer uniquement les évolutions entre deux démarrages du système. Un changement est pris en compte à condition de redémarrer le système. La complexité est plus faible car le logiciel se configure lors de sa phase d'initialisation, et cette configuration n'est plus changée durant l'exécution. Cette solution n'est pas envisageable pour gérer le dynamisme d'un environnement pervasif : en plus d'être contraignante pour l'utilisateur et contraire à la disparition de l'informatique de sa conscience, le système global est composé d'un ensemble de sous-systèmes qu'il faudrait potentiellement redémarrer, éventuellement dans un ordre particulier. La gestion du dynamisme pendant l'exécution est plus compliquée, mais elle est souhaitable dans le cadre de l'informatique pervasive. Par rapport à l'adaptation statique, la difficulté provient de la volonté de maintenir au mieux le fonctionnement du système pendant que certaines parties sont modifiées. Deux problèmes peuvent émerger : la préservation d'un état courant des variables d'une part, et du flot d'exécution d'autre part. Selon la méthode d'adaptation d'un système, l'état courant et le flot d'exécution n'est pas impacté. Si le système est considéré comme critique, cette préservation est primordiale.

Une première méthode d'adaptation est de coder l'adaptation dans le code source. Dans un langage de programmation impératif [20], c'est typiquement le branchement conditionnel qui permet d'adapter le flot d'exécution aux évolutions. Deux principaux problèmes se posent avec cette méthode d'adaptation pour les environnements pervasifs. Tout d'abord, le très grand nombre de cas possibles la rend peu praticable. Ensuite, l'adaptation étant écrite une fois pour toutes dans le code, toutes les adaptations possibles doivent être connues lors de la conception, ce qui n'est pas le cas en général pour les environnements pervasifs. Cette méthode ne pose pas de problème pour la préservation de l'état courant et du flot d'exécution. Une deuxième solution est de remplacer une partie du code par un autre code, ou d'ajouter du code. Cette solution s'appelle le *hot swapping* [21]. La capacité à faire du hot swapping peut être fournie de base par le langage de programmation, ou par un outil fourni pour un langage donné. Par exemple, le langage purement objet Smalltalk [Callaú *et al.*, 2011], et le langage fonctionnel Erlang [Primi, 2007] incluent la capacité de hot swapping. Java permet de faire du hot swapping seulement en mode débogage [22], cette fonctionnalité est donc destinée aux environnements de développement plutôt qu'aux environnements de production. Du fait de l'absence d'effet de bord des fonctions, le remplacement d'une fonction par une autre dans un langage purement fonctionnel n'impacte pas l'état courant. Les langages objets au contraire permettent aux objets d'avoir un état interne. Ainsi, le remplacement d'un objet par un autre n'est pas aisé si l'on souhaite conserver l'état de l'objet remplacé. Enfin, l'architecture du système peut permettre de faire de l'adaptation dynamique, si elle est conçue pour. Par exemple, le modèle d'architecture à service permet un couplage faible entre les différentes parties d'un système, ce qui permet de remplacer des parties sans impacter les autres.

2.2 Interfaces multimodales

L'ensemble composé des dispositifs d'entrée et de sortie, ainsi que toutes les entités qui effectuent des traitements sur les données qui transitent jusqu'au noyau fonctionnel, constitue une **interface utilisateur**. Du point de vue de l'utilisateur, l'interface utilisateur est constituée des dispositifs d'entrée et sortie, ainsi que du langage d'interaction à employer pour communiquer

20. Les langages de programmation généralistes les plus utilisés sont impératifs : C, Java, C++ (voir `http://langpop.com/`).

21. *Hot swapping* peut être traduit par « échange à chaud ».

22. Voir la page *JPDA Enhancements* : `http://docs.oracle.com/javase/1.4.2/docs/guide/jpda/enhancements.html`. Pour être précis, c'est l'API standard fournie par le debugger qui permet de faire du hot swapping, plutôt qu'une fonctionnalité du langage.

avec le noyau fonctionnel.

2.2.1 Dispositif d'interaction

Pour exploiter les capacités d'un système informatique pour réaliser une tâche, tout utilisateur doit exprimer ses besoins. Plus généralement, il doit transmettre de l'information compréhensible par un système informatique. Pour cela, il code l'information en dépensant de l'énergie sous une certaine forme. Une partie de cette énergie est captée par des transducteurs : appui sur un bouton convertissant l'énergie de pression en énergie électrique, vibration de l'air avec les cordes vocales qui est captée par un microphone convertissant la pression de l'air en signal électrique, etc. Les transducteurs peuvent être couplés à une partie logicielle chargée d'extraire des informations utiles : le signal électrique du microphone est généralement analysé pour être converti en une suite de mots. Le dispositif qui contient un transducteur, éventuellement couplé avec un dispositif logiciel, est le dispositif d'entrée. Le transducteur du dispositif d'entrée peut être considéré comme la frontière entre le système informatique et l'environnement physique. L'information fournie par l'utilisateur transite au sein du système informatique jusqu'au noyau fonctionnel [23] qui réalise effectivement le service. Dans l'autre sens, le système fournit une information perceptible par l'utilisateur. Cette information transite au sein du système informatique jusqu'à un transducteur chargé de rendre perceptible l'information par un humain. Ce transducteur, éventuellement couplé à un dispositif logiciel, est le dispositif de sortie. Il existe une grande variété de dispositifs d'interaction en entrée et en sortie, comme souligné à la section 2.1.1.2. De plus, des taxonomies permettent de caractériser finement les dispositifs en entrée [24].

Du point de vue de l'utilisateur, l'interface utilisateur est constituée des dispositifs d'entrée et de sortie, ainsi que des langages d'interaction utilisés par l'utilisateur pour communiquer avec le noyau fonctionnel et des langages d'interaction utilisés par le noyau fonctionnel pour rendre perceptibles son état interne et ainsi les effets des actions de l'utilisateur.

Dans nos travaux, nous traitons de l'interaction multimodale en entrée, de l'utilisateur vers le noyau fonctionnel. Aussi dans les sections suivantes, nous mettons l'accent sur les dispositifs et les langages d'interaction en entrée.

2.2.2 Langage d'interaction

Un langage d'interaction est un ensemble de conventions défini par l'interface utilisateur qui permet d'échanger des informations entre un utilisateur et un noyau fonctionnel. Ces conventions sont généralement empruntées aux interfaces utilisateur préexistantes pour ce qui est des fonctions communes à la plupart des logiciels : par exemple, un clavier d'ordinateur permet de couper ou copier un objet qui a le focus [25] avec les touches x, c et la touche *control*. Un tel langage peut être appelé un langage de commandes. Pour les conventions propres au type de service fourni par le noyau fonctionnel, elles sont aussi empruntées à des logiciels de même type qui existent déjà. La touche *espace*, par exemple, permet la mise en pause dans beaucoup de logiciels de type lecteur de médias. Beaucoup de conventions existent pour l'utilisation du clavier et de la souris ; ces conventions s'appliquent à l'échelle du système d'exploitation. Les boites à outils utilisables par les développeurs d'interfaces graphiques procurent

23. Dans le domaine de l'IHM, le noyau fonctionnel correspond à la partie métier, c'est-à-dire la partie logicielle qui réalise effectivement le service fonctionnel que requiert l'utilisateur.

24. Voir la section 2.1.1.2.1 page 23.

25. Dans une interface graphique, l'objet graphique qui a le focus est celui qui sur lequel s'applique les entrées. En général, un objet qui a le focus est distingué des autres : une section de texte est surlignée, un bouton est encadré, etc.

un ensemble de facilités et de comportements par défaut qui permettent de suivre ces conventions. Par exemple, si un développeur définit un champ texte modifiable fourni par une boite à outils, alors ce champ bénéficie de la possibilité de copier et coller du texte ou d'annuler les modifications du champ avec les raccourcis clavier standards ou avec un menu contextuel. Si la boite à outils est multiplateforme, alors les raccourcis clavier seront adaptés aux conventions de la plateforme d'exécution. Si les langages d'interaction utilisés avec la souris et le clavier bénéficient d'un ensemble de conventions maintenant bien établies, il n'en est pas de même pour beaucoup d'autres dispositifs d'interaction. En effet, l'utilisation par tous les utilisateurs du clavier et de la souris depuis les années 1980 a vraisemblablement poussé les éditeurs de logiciels et les constructeurs matériels à offrir des langages d'interaction identiques, ou similaires. Les nouveaux dispositifs permettent de créer de nouveaux langages d'interaction sans affronter la résistance au changement des utilisateurs.

Un langage d'interaction en entrée est par définition dépendant du dispositif d'interaction qui fournit les données de bas niveau qui sont abstraites selon le langage en information. De plus, le langage d'interaction est dépendant du noyau fonctionnel et des tâches ou commandes qu'il fournit puisque ces tâches définissent la sémantique des informations liées au langage.

Enfin, un langage d'interaction est destiné à être employé par un utilisateur. Il se doit donc d'être en adéquation avec la représentation mentale d'un utilisateur de la tâche à accomplir, tout en étant facile à apprendre et à retenir. Par exemple, la commande *copier* – *copy* en anglais – correspond à l'appui de la touche c et d'une touche activant le mode commande. Les commandes avec une sémantique proche, *couper* et *coller*, correspondent à l'appui des touches x et v, ces trois touches sont adjacentes sur la plupart des dispositions claviers. Ce regroupement spatial a vraisemblablement été guidé par l'envie de regrouper les commandes aux sémantiques proches pour ainsi faciliter la mémorisation par l'utilisateur du langage de commande.

En synthèse, un langage d'interaction est un ensemble de conventions entre l'utilisateur et un noyau fonctionnel qui permet l'échange d'informations entre eux. Un langage d'interaction est dépendant d'un noyau fonctionnel et du dispositif qui permet à l'utilisateur de s'exprimer dans ce langage.

2.2.3 Modalité d'interaction

Une modalité d'interaction en entrée est une modalité qui permet de véhiculer l'information d'un utilisateur vers un noyau fonctionnel. À l'opposé, une modalité en sortie est une modalité qui permet de véhiculer de l'information d'un noyau fonctionnel vers un utilisateur.

Dans la littérature scientifique du domaine de l'IHM, le terme *modalité d'interaction* peut être remplacé par le terme *technique d'interaction*. Néanmoins le terme technique d'interaction est aussi utilisé pour désigner des objets graphiques interactifs. Cet usage du terme se réfère à l'interaction instrumentale [Beaudouin-Lafon, 2000] qui explicite la notion d'instrument logique ou dispositif logique.

De plus le terme modalité d'interaction est systématiquement utilisé dès qu'il y a un mécanisme de reconnaissance mis en œuvre et donc incertitude dans l'interprétation des données issues des dispositifs. Par exemple, il est classique d'étudier la modalité parole, la modalité gestuelle.

De nombreuses définitions du terme *modalité d'interaction* ont été proposées. Plusieurs définitions sont recensées dans [Nigay et Coutaz, 1996], et une proposition de définition est donnée :

> **Une modalité est l'association d'un dispositif d'interaction et d'un langage d'interaction**.

Par exemple l'association d'un clavier et d'un langage de commande, peut définir la modalité basée sur les raccourcis claviers, qui a été utilisée dans les exemples précédents. Dans la suite

de ce document nous adhérons à cette définition car elle permet d'adopter à la fois le point de vue technique (développement d'une modalité) et le point de vue de l'utilisateur (usage de la modalité).

2.2.4 Multimodalité

2.2.4.1 Définition

La multimodalité caractérise une interface qui intègre plusieurs modalités en entrée ou en sortie. Les interfaces multimodales sont considérées comme une classe d'interface hommes machines à part entière, au même titre que les interfaces WIMP [26] [Dumas *et al.*, 2009]. En 1980, Richard A. Bolt, avec son article « "Put-That-There": Voice and Gesture Interface at the Graphics Interface » [27] est reconnu comme le premier auteur à souligner les avantages à utiliser deux modalités complémentaires pour interagir avec un système interactif [Bolt, 1980]. Dans cet article, il décrit un système où plusieurs modalités d'entrée sont utilisées pour réaliser une tâche : le « mets ça là » [28]. Pour cela, l'utilisateur décrit l'action à réaliser avec la voix en énonçant « mets », puis montre la forme à déplacer avec un geste en énonçant simultanément « ça », et enfin désigne le nouvel emplacement en énonçant simultanément « là ».

Deux principaux bénéfices sont attendus des interfaces multimodales [Oviatt, 1999] :

- Offrir un moyen d'interaction plus adapté aux tâches et à l'utilisateur. En choisissant une modalité, la satisfaction de l'utilisateur peut être accrue. En utilisant une modalité adaptée à la tâche, le nombre d'erreurs peut être minimisé.
- Rendre plus robuste l'interprétation des informations issues de l'utilisateur. Actuellement, les mécanismes de reconnaissance de gestes, de parole, etc, ne sont pas capables d'effectuer une reconnaissance correcte en toute situation. L'usage d'une autre modalité en parallèle peut augmenter le taux de reconnaissance correcte.

Sharon Oviatt a mené une expérience dans le but d'évaluer les différences entre interfaces multimodales et unimodales [Oviatt, 1997]. Cette évaluation porte sur l'utilisation d'une carte interactive en deux dimensions en utilisant la parole, le pointage avec un stylet, et les deux. De cette évaluation, il ressort que les utilisateurs font moins d'erreurs avec une interface multimodale, et qu'ils préfèrent quasiment tous les interfaces multimodales. De plus, leur efficacité est légèrement améliorée. Il n'est néanmoins pas évident que ce résultat soit généralisable à toutes les interfaces multimodales.

Les interfaces multimodales sont souvent considérées comme naturelles parce qu'elles intègrent des modalités dites naturelles, comme les interfaces vocales [Juang et Rabiner, 2004], les modalités gestuelles [Mitra et Acharya, 2007] ou encore les modalités basées sur l'utilisation d'un stylet électronique [Dai, 2004]. Cet emploi du terme « naturel » est à prendre avec précaution comme le souligne Norman [Norman, 2010]. Le caractère naturel fréquemment associé aux interfaces multimodales est aussi issu de l'exemple séminal du « mets ça là » de Bolt.

D'un point de vue de l'utilisation, la multimodalité permet à l'utilisateur soit de **choisir** la modalité qu'il ou elle préfère, soit d'**utiliser conjointement** plusieurs modalités, de façon séquentielle ou parallèle. Ces deux possibilités d'utilisation de la multimodalité ne sont bien sûr pas exclusives. Elles sont décrites plus en détail avec les relations [Nigay et Coutaz, 1997].

La suite se concentre sur les interfaces multimodales en entrée.

26. WIMP est l'acronyme de *windows, icons, menus and pointer*, c'est-à-dire « fenêtres, icones, menus et curseur », le style d'interaction très utilisé pour les interfaces graphiques.
27. En français : *« Mets ça là » : interface vocale et gestuelle pour une interface graphique.*
28. En anglais : *put that there.*

2.2.4.2 Choix de modalités : équivalence

Le choix par l'utilisateur d'une modalité peut être motivé de plusieurs manières : en fonction de ses habitudes, de ses envies, de son habileté à utiliser telle ou telle modalité, de sa proximité spatiale avec un dispositif, de l'adéquation de la modalité à la tâche qu'il souhaite effectuer, de l'environnement physique, etc. Donner le choix à l'utilisateur entre le clavier et la souris pour réaliser toute tâche est une recommandation pour le développement d'interface utilisateur [Berry, 1988]. Transposé aux environnements pervasifs, un concepteur d'interfaces utilisateur devrait, pour chaque tâche, donner le choix à l'utilisateur entre toutes les modalités existantes. Bien sûr, cette recommandation n'est valable que si une modalité considérée permet de fournir une entrée attendue par le noyau fonctionnel pour une certaine tâche. Cependant, les concepteurs doivent éviter de faire des suppositions sur les préférences de l'utilisateur, puisque le choix n'est pas uniquement guidé par l'adéquation d'une modalité à la tâche à effectuer. De plus, deux environnements pervasifs n'ont pas nécessairement les même modalités disponibles, il est donc malaisé de présupposer la présence d'une certaine modalité.

L'**équivalence** est une des quatre propriétés CARE [29]. L'**assignation** – autre propriété CARE – est la restriction d'une tâche à une modalité : cette tâche de doit pas pouvoir être réalisée par une autre modalité. Elle caractérise l'absence de choix pour l'utilisateur. Cette propriété va à l'encontre de caractéristiques des environnements pervasifs : dynamisme (l'unique modalité peut disparaitre, la tâche devient alors inaccessible ; une nouvelle modalité apparaissant peut constituer une modalité plus adaptée), et état de l'environnement à l'exécution inconnu lors de la spécification.

Les deux propriétés CARE restantes sont abordées dans la section suivante.

2.2.4.3 Combinaison de modalités : complémentarité et redondance

Utiliser conjointement plusieurs modalités signifie que l'interface multimodale est capable de corréler ou combiner des informations issues de plusieurs modalités. L'exemple de Bolt combine l'information issue de la modalité parole – la fonction voulue par l'utilisateur – avec les informations issues de la modalité gestuelle – les objets sur lesquels porte la fonction. L'information complète reçue par le noyau fonctionnel est produite à partir des informations qui sont indépendantes lors de leur entrée dans l'interface utilisateur. La corrélation d'informations issues de plusieurs modalités correspond à leur mise ensemble dans le but d'interpréter plus sûrement leur sémantique. Par exemple, si un utilisateur énonce « oui » tout en hochant d'approbation la tête, alors le système peut interpréter plus sûrement le consentement de l'utilisateur.

La corrélation d'informations issues de plusieurs modalités est intéressante pour les modalités intégrant un mécanisme de reconnaissance, comme la reconnaissance de la parole, où un mot reconnu peut être annoté avec un degré de confiance par la partie chargée de la reconnaissance. Une information issue d'une autre modalité peut faire varier ce degré de confiance. Un degré de confiance vise à guider l'interface multimodale dans son choix de l'interprétation. La corrélation d'information est également très intéressante dans le cas de tâches critiques, comme la suppression de fichiers : deux modalités peuvent être requises pour effectuer la tâche, et ainsi éviter les activations accidentelles de la tâche.

L'intérêt de la combinaison d'informations dépend largement du noyau fonctionnel. Les bonnes pratiques en conception d'interface utilisateur recommandent d'utiliser une architecture qui découple le noyau fonctionnel de la partie interface, c'est une application du principe de la séparation des préoccupations du génie logiciel. C'est ce que proposent les modèles d'architecture comme MVC [Krasner et Pope, 1988], Arch [archSigchiBull, 1992], PAC [Coutaz, 1987],

29. Acronyme de *complementarity*, *assignment*, *redundancy*, *equivalence*, soit complémentarité, assignation, redondance et équivalence.

etc. Ainsi, le noyau propose un ensemble de tâches qui peuvent être réalisées par un utilisateur. L'interface en entrée interprète les informations produites par l'utilisateur pour finalement faire exécuter par le noyau fonctionnel la tâche souhaitée par l'utilisateur. Faire exécuter une tâche par le noyau fonctionnel peut être modélisé comme un appel de fonction qui a éventuellement des paramètres. Le noyau fonctionnel est alors une boite noire qui dispose d'une API [30]. Donc, le rôle d'une interface utilisateur est de trouver quelle fonction appeler, et quels paramètres lui fournir. Dans un premier cas simple de combinaison, la fonction à appeler et les paramètres sont directement récupérés des données fournies par l'utilisateur. Les cas plus compliqués émergent du type des paramètres : une fonction de création de formes dans un logiciel de dessin peut requérir un point, c'est-à-dire un couple de valeurs qui représente une coordonnée. Ces valeurs peuvent être fournies par deux modalités en entrée distinctes et doivent être combinées pour former une coordonnée.

2.2.4.4 Synthèse

En synthèse, nous considérons deux niveaux d'abstraction dans une modalité d'interaction : le dispositif et le langage. Nous retenons de la multimodalité ses deux apports principaux que sont la flexibilité en offrant des choix aux utilisateurs et la possibilité d'utiliser conjointement plusieurs modalités.

2.3 Conclusion : pourquoi la multimodalité en environnement pervasif ?

Nous concluons ce chapitre en présentant l'apport de la multimodalité pour les environnements pervasifs, puis une étude préliminaire sur la manière d'intégrer les interfaces multimodales dans les applications est présentée.

2.3.1 Apport de la multimodalité pour les environnements pervasifs

L'hétérogénéité des dispositifs présents dans les environnements pervasifs est inhérente aux environnements pervasifs. Ainsi, une interface utilisateur dans un environnement pervasif qui ne pose pas de prérequis fort sur les dispositifs d'interaction est par définition une interface multimodale, puisque plusieurs dispositifs – donc plusieurs modalités – peuvent être utilisées. La multimodalité peut-être ainsi vue comme une composante inhérente des interfaces utilisateur en environnement pervasif.

Par ailleurs, le choix donné aux utilisateurs finaux par les interfaces multimodales est une caractéristique intéressante pour les environnements pervasifs : elles améliorent l'opportunisme des interfaces utilisateur.

Enfin, le dynamisme est une caractéristique des environnements pervasifs, notamment les arrivées et départs de dispositifs d'interaction. La multimodalité, en s'intéressant à la combinaison de modalités, permet d'exploiter différemment un même dispositif selon les autres dispositifs et les services présents dans l'environnement à un instant donné.

La multimodalité augmente donc la flexibilité et l'opportunisme des interfaces utilisateur.

30. API est le sigle d'*advanced programming interface*, qui peut se traduire par « interface de programmation ».

2.3.2 Greffe de multimodalité : notre hypothèse

Alors que la plupart des applications proposent une interface graphique, seule une partie infime est conçue pour aller au-delà de la multimodalité proposée de base par les boites à outils graphiques. Il est illusoire de croire que les concepteurs vont demain se mettre à développer des interfaces multimodales qui ne sont pas déjà gérées par ces boites à outils. Quelles solutions existent pour ajouter une interface multimodale en entrée à une application existante et à celles à venir ?

Une interface multimodale en entrée se situe entre un utilisateur et un noyau fonctionnel. L'immense majorité des applications interactives sont dotées d'une interface utilisateur graphique en sortie et l'interaction en entrée repose sur la manipulation directe. Dans ce contexte, il est très difficile de découpler clairement l'interface en entrée de l'interface en sortie. Par exemple le curseur à l'écran qui reflète l'endroit actuellement pointé par l'utilisateur est un concept de l'interface graphique en sortie. De même dans l'exemple de Bolt, l'interface multimodale en entrée est dépendante de l'interface graphique en sortie. En effet le pointage se fait sur un point de l'écran d'affichage, et c'est l'interface graphique en sortie qui peut faire la correspondance entre ce point et un objet du noyau fonctionnel. Si ce couplage des interfaces en entrée et en sortie n'est pas un problème lorsque l'interface multimodale en entrée est développée en même temps que l'interface graphique en sortie, c'est-à-dire que le code de chaque interface peut être modifié en fonction de l'autre, il devient problématique si un concepteur souhaite ajouter une interface multimodale en entrée à un logiciel déjà composé d'un noyau fonctionnel et de son interface graphique en sortie. Ce cas apparaît presque systématiquement dans l'informatique pervasive pour les logiciels existants : ils n'ont pas été conçus pour être utilisés par une interface multimodale en entrée. Le logiciel est alors une seule unité, l'interface graphique en sortie et le noyau fonctionnel ne sont pas dissociables, vu de l'extérieur. Même si son architecture respecte le couplage faible entre le noyau fonctionnel et l'interface graphique, cette architecture est rarement accessible de l'extérieur. Par contre, un logiciel peut proposer une API pour être manipulable de l'extérieur.

Une approche interne, pragmatique, serait de modifier le logiciel même. Cependant, cette approche est plutôt limitée. Si c'est le concepteur de logiciel qui souhaite ajouter une interface multimodale, il peut modifier le code source. Mais comme vu dans la partie hétérogénéité, il est très compliqué de faire une interface susceptible de prendre en compte tous dispositifs, donc toutes les modalités existantes. De plus, si l'extension multimodale n'est pas faite par le concepteur lui-même, l'accès au code source est nécessaire pour faire des modifications. Or cet accès au code source n'est pas systématique. De plus modifier du code sans qu'il soit intégré dans le projet initial demande de gros efforts de maintenance pour suivre l'évolution du projet : il est nécessaire de fusionner les évolutions de code au fur à mesure, ce qui peut être compliqué, et les utilisateurs finaux n'ont pas forcément connaissance du logiciel résultant dérivé. De plus, ajouter une interface multimodale à un logiciel n'est pas forcément une caractéristique importante qui amène un utilisateur à choisir un logiciel dérivé plutôt que l'original. Le travail autour du logiciel est dédoublé : documentation, rapports de bugs, site web, communauté d'utilisateurs, etc.

Aussi, l'approche externe semble réalisable avec beaucoup moins d'efforts. L'inconvénient majeur de cette approche est l'impossibilité d'accéder à l'état interne du logiciel, et notamment à l'état de l'interface graphique en sortie. Cet inconvénient est à relativiser par les bonnes pratiques de développement : un noyau fonctionnel doit masquer son état interne et n'être accessible que par une API. Ainsi, la facilité d'appliquer une approche externe est très dépendante de la qualité et de l'étendue de la couverture de l'API d'un logiciel. L'approche externe suppose la possibilité d'étendre un logiciel via une API. S'il est difficile d'avoir une vision globale de tous les logiciels actuels, une tendance des logiciels à proposer des mécanismes d'extension peut être remarquée.

Par exemple, les navigateurs web comme Mozilla Firefox [31], Google Chrome [32] proposent des mécanismes d'extension. De même que les environnements de développement comme Eclipse [33] et NetBeans [34], les applications web comme MediaWiki [35], les services en ligne comme Google Earth [36] et Facebook [37], etc. L'approche externe semble donc prometteuse et surtout pragmatique dans le contexte actuel.

Partant de la vision de Weiser qui a vingt ans, les concepts sont établis. En nous reposant sur ce socle conceptuel, nos travaux sont consacrés aux aspects d'ingénierie des environnements pervasifs. Aussi le chapitre suivant présente les outils et plateformes existantes qui traitent des environnements pervasifs et de l'interaction multimodale.

31. Site officiel : `http://www.mozilla.org/en-US/firefox/new/`, site d'entensions : `https://addons.mozilla.org/en-US/firefox/`.
32. Site officiel : `https://www.google.com/chrome`, site d'extensions : `https://chrome.google.com/extensions`.
33. Site officiel : `http://www.eclipse.org/`.
34. Site officiel : `http://netbeans.org/`.
35. Site officiel : `http://www.mediawiki.org/`, liste d'extensions : `http://www.mediawiki.org/wiki/Special:AllPages/Extension:`.
36. Site officiel : `https://developers.google.com/earth/`, API : `http://code.google.com/apis/earth/`.
37. Site officiel : `http://www.facebook.com/`, site pour les développeurs : `http://developers.facebook.com/`.

Chapitre 3

Plateformes logicielles pour l'informatique pervasive et l'interaction multimodale

Dans le chapitre précédent, nous nous sommes concentrés sur les concepts clefs de l'informatique pervasif et de la multimodalité et nous avons montré l'intérêt de l'interaction multimodale dans un environnement pervasif. Les éléments conceptuels étant fixés, nous consacrons le reste de ce mémoire à la réalisation logicielle de systèmes multimodaux pervasifs. Pour cela, nous étudions dans ce chapitre les outils existants puis nous présentons dans le chapitre 4 notre solution conceptuelle pour la réalisation logicielle avant de présenter une implémentation de notre solution, la plateforme DynaMo au chapitre 5.

Dans ce chapitre nous organisons notre revue des outils logiciels en deux parties. Nous étudions d'abord les plateformes généralistes pour gérer le dynamisme, notamment celles axées sur l'informatique pervasif. Puis nous étudions les plateformes qui dédiés à la multimodalité. Notre revue des plateformes logicielles couvrent donc au moins deux domaines de recherche : le génie logiciel et l'interaction homme-machine.

L'hétérogénéité – abordée dans le chapitre précédent – ne fait pas l'objet d'une section particulière dans ce chapitre. En effet, le concept d'hétérogénéité est très vaste et concerne tous les domaines, il est donc traité de manière transverale tout au long de ce chapitre.

3.1 Plateformes généralistes pour le dynamisme

Dans la section 2.1.2 a été abordé le dynamisme dans les environnements pervasifs. En logiciel, gérer le dynamisme implique de connaitre l'évolution de l'état de l'environnement et d'adapter le logiciel en conséquence. L'adaptation d'un système implique la gestion du dynamisme. Le but de cette section est de présenter des travaux qui offrent des mécanismes pour adapter des systèmes.

Cette section est divisée en deux grandes parties : elle présente d'une part des plateformes généralistes pour le dynamisme et d'autre part des plateformes plus spécialisées pour la gestion du dynamisme au sein des environnements pervasifs.

3.1.1 Introduction

Quel que soit le niveau d'abstraction utilisé pour gérer le dynamisme, il repose en dernier lieu sur les mécanismes de bas niveau fourni par le matériel électronique. Ainsi, les possibilités offertes à des plus hauts niveaux d'abstraction ne peuvent dépasser celles du matériel. Quel que soit le niveau d'abstraction proposé à un concepteur, il existe donc une limite. Comparer les possibilités de gestion du dynamisme s'entend comme la comparaison de la facilité de développement, de la rapidité, et éventuellement des restrictions imposées.

Plusieurs langages généralistes proposent des mécanismes pour le dynamisme. Cependant, ces mécanismes ne portent que sur les concepts proposés par le langage. Ces concepts sont d'assez bas niveau au regard de l'architecture globale d'un système créé avec ce langage : la granularité prise est bien souvent trop petite pour concevoir un système globalement dynamique reposant uniquement sur ces mécanismes. Aussi, la prise en compte du dynamisme peut être relayée au niveau de l'architecture d'un système. Deux approches architecturales proposent des mécanismes dynamiques de plus gros grains : l'approche à composant et l'approche à service.

L'**approche à composant** propose de construire des applications en assemblant des composants. Plusieurs définitions du concept de composant existent [Lau et Wang, 2005]. Szyperski définit un composant comme « une unité de composition avec des interfaces spécifiées par contrat et des dépendances contextuelles explicites uniquement. Un composant peut être déployé de façon indépendante, et il est sujet à la composition par des tiers »[Szyperski *et al.*, 2002, p. 41] [1]. Le concept de composition est primordial dans l'approche à composant. Il suppose la présence d'un langage de composition. Une composition est un assemblage d'instances de composants. Une instance de composant est une copie d'un composant qui possède un état et peut être paramétré. Selon le contexte, le terme composant peut faire référence à une instance de composant. Un composant est représenté dans la figure 3.1.

L'approche à composants promeut la composition de briques de base génériques. Ainsi, la réutilisation se fait à un grain plus fin que les logiciels complets génériques. Les mises à jour peuvent se faire au niveau du composant et non du logiciel, limitant ainsi leurs effets potentiellement négatifs. L'utilisation d'une technologie ouverte à composant permet la pluralité des fournisseurs de composants : un composant est défini par son interface, permettant notamment de définir une équivalence entre des composants. Le concept de composant tel que présenté ici est très général, les technologies fournissant le concept de composant sont appelées des modèles à composants. Un modèle à composant définit trois aspects : la syntaxe d'un composant, la sémantique d'un composant, et la manière de les assembler [Lau et Wang, 2005].

Le concept de conteneur – ou membrane – est bien souvent proposé par les modèles à composants. Il permet d'encapsuler un noyau de composant, correspondant à la sémantique du composant, par des caractéristiques générales à tous les composants. Un conteneur peut par exemple fournir des mécanismes d'arrêt et de démarrage d'un composant. Pour permettre de spécialiser les instances de composant, un modèle à composant peut permettre à un composant d'être exprimé en fonction d'une configuration, c'est-à-dire exprimer l'algorithme d'un composant en fonction de variables qui prennent une valeur lors de l'instanciation. La configuration d'un composant peut être une manière de faire de l'adaptation au niveau d'un composant. La gestion du dynamisme n'est pas un présupposé de l'approche à composant, mais elle est proposée par certains modèles à composants, qui proposent alors différents mécanismes dynamiques.

L'approche à service est un paradigme qui utilise des services comme briques de base pour le développement d'application [Papazoglou et Georgakopoulos, 2003]. Un service est une entité

1. Proposition de traduction pour « a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. ».

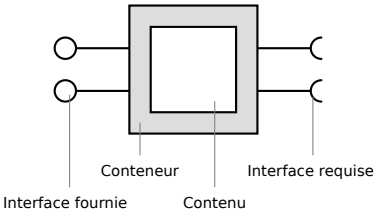


FIGURE 3.1 – Représentation d'un composant.

autonome qui est composée d'une description de ses capacités, et d'une entité logicielle qui réalise effectivement ces capacités. Une application est alors un ensemble de services sélectionnés, puis composés. Ce paradigme vise à faciliter la création d'**applications flexibles et dynamiques**, en promouvant le couplage faible entre les différentes parties d'une application et la possibilité de sélectionner des services à l'exécution.

L'**approche orientée service** repose sur l'architecture orientée service. Cette architecture est composée de quatre éléments : une spécification de service, un fournisseur de service, un consommateur de service et un annuaire de services. L'objectif du consommateur de service est d'utiliser un service dont il a besoin. Cette utilisation se fait en trois étapes. D'abord, la **publication** : le fournisseur s'enregistre dans l'annuaire en lui donnant sa spécification, c'est-à-dire la description de ses capacités et un ensemble de propriétés qu'il possède. Ensuite, la **découverte** : le consommateur demande à l'annuaire l'ensemble des services qui sont susceptibles de répondre à son besoin. Enfin, la **liaison** : le consommateur sélectionne un fournisseur et se connecte à lui pour l'utiliser. Cette architecture est représentée dans la figure 3.2.

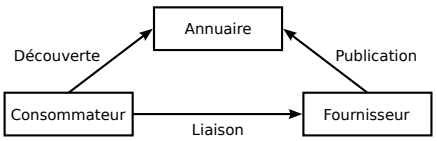


FIGURE 3.2 – Architecture orientée service.

Alors que l'approche à composant ne précise pas les mécanismes de liaison entre composants, le mécanisme de liaison de services en trois étapes proposé par l'architecture orientée service facilite la liaison retardée [2], ouvrant ainsi clairement la voie à une architecture dynamique. La spécification de service, comme les interfaces des composants, permet d'introduire une certaine équivalence entre les entités.

L'approche à composant, comme l'approche à service, donne des principes architecturaux pour construire des applications, d'un niveau d'abstraction plus haut que les concepts des langages de programmation généralistes. Ces deux approches donnent un cadre et des outils qui peuvent faciliter et accélérer la conception d'applications, mais aussi fournir des mécanismes dynamiques robustes qui sont pertinents pour la construction de systèmes pervasifs. Nous illustrons ces deux points dans les sections suivantes.

2. Traduction de *late binding*.

3.1.2 OSGi

OSGi est une spécification ouverte définie par l'OSGi Alliance, un consortium mondial fondé en 1999, qui vise à définir une plateforme pour la construction d'applications modulaires en Java, dans le but de réduire la complexité logicielle [3]. En 2011, le consortium est notamment composé de géants du secteur informatique comme Hitachi, IBM, NTT, Oracle, Siemens AG, etc. Ce standard suit l'approche à service.

OSGi est donc une spécification définissant une plateforme qui permet d'utiliser l'approche à service. Cette spécification repose sur le langage Java. Elle définit une manière de développer, déployer et gérer des services [osgi43, 2011]. Elle se concentre en premier lieu sur une utilisation locale [4] de l'approche à service, bien qu'elle spécifie une architecture de haut niveau pour gérer la distribution. Elle est divisée en plusieurs couches, dont la couche module, la couche cycle de vie et la couche service. La couche service repose sur la couche cycle de vie, qui repose sur la couche module.

La **couche module** définit une modularisation selon des entités appelées *bundles*, qui sont une mise en commun de ressources et de codes exécutables [5]. Un bundle peut dépendre d'entités externes : un autre bundle, une bibliothèque partagée, un périphérique, etc. Il peut également déclarer les capacités qu'il fournit, desquelles un autre bundle peut dépendre. Cette modularisation peut être vue comme une approche à composant.

La **couche cycle de vie** définit un bundle comme une unité de déploiement, c'est-à-dire qu'il peut être installé, démarré, mis à jour, arrêté et désinstallé. L'installation se fait au sein d'une plateforme OSGi qui prend en charge le cycle de vie des bundles. Le standard définit une API pour agir sur le cycle de vie ; un bundle ou un administrateur via une interface utilisateur peut utiliser cette API. Les dépendances doivent être satisfaites pour que le bundle puisse démarrer. Lorsqu'un bundle démarre, il passe par une phase d'activation qui lui permet de s'initialiser : allocation de ressources, création de processus, enregistrement de services, etc. De même, lorsqu'il s'arrête, il doit libérer les ressources, arrêter les processus démarrés, etc. L'ensemble des états possibles d'un bundle et des transitions possibles entre les états est résumé dans la figure 3.3.

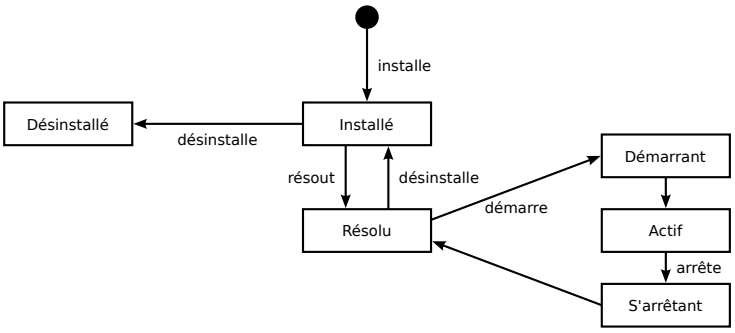


FIGURE 3.3 – Cycle de vie d'un bundle : enchainement possible des états d'un bundle. Adaptée de [osgi43, 2011, p. 90].

Enfin, la **couche service** est dédiée à l'approche orientée service. Cette couche utilise la couche cycle de vie pour suivre l'évolution de la disponibilité des services. C'est un bundle qui propose ou requiert des services. Lorsqu'il propose un service, c'est au bundle de l'enregistrer

3. Site officiel du consortium : `http://www.osgi.org/`.
4. Local signifie ici au sein d'une même plateforme, sur une seule machine.
5. En Java, le code exécutable est regroupé en classes, sous forme de bytecode compréhensible par une machine virtuelle Java.

dans l'annuaire ou de le désenregistrer. L'aspect dynamique d'OSGi permet au bundle de faire ces opérations à n'importe quel moment. Lors de l'enregistrement d'un service, le bundle fournit l'interface Java [6] implémentée par le service, et un ensemble de propriétés que ce service respecte. L'interface Java et l'ensemble de propriétés correspondent à la spécification de service de l'architecture orientée service. Les propriétés sont importantes car elles constituent les critères qui permettent à un consommateur de service de sélectionner un service parmi l'ensemble des services qui correspondent à son besoin. Si un bundle a besoin d'un service, il peut s'enregistrer pour être prévenu des évolutions de la disponibilité de ce service. C'est au bundle d'effectuer le travail nécessaire pour tenir compte de ces évolutions. Par contre, puisque la plateforme OSGi gère le cycle de vie des bundles, elle désenregistre automatiquement les services proposés par un bundle lorsque celui-ci s'arrête.

Par l'intégration de la gestion du cycle de vie de bundles avec les mécanismes de l'approche à service, OSGi permet de développer des applications dynamiques. Cependant, les concepteurs de bundles doivent gérer eux-mêmes les apparitions et disparitions de services, qui peuvent intervenir à tout moment lors de l'exécution. C'est une tâche complexe et source d'erreurs, notamment parce que un bundle fournit du code métier et du code qui gère la disponibilité des services : le principe de séparation des préoccupations n'est pas respecté. De plus, OSGi ne fournit pas une vue de haut niveau d'abstraction qui permet de composer les services entre eux.

3.1.3 iPOJO

iPOJO est un modèle à composants orientés services. Il a été réalisé dans le cadre d'une recherche doctorale au sein de l'équipe Adèle du laboratoire d'informatique de Grenoble. La thèse a été soutenue en 2008 [Escoffier, 2008]. iPOJO constitue un sous projet d'Apache Felix, l'implémentation d'OSGi de la fondation Apache, il est encore développé en 2012.

L'idée des **composants orientés service** a émergé du constat qu'il est difficile de réutiliser ou changer des compositions de services [Yang, 2003], dû au manque de structuration des compositions que permettent de produire les outils existants. De plus, il est impossible de spécifier des compositions structurelles avec l'approche orientée service [Cervantes et Hall, 2004]. L'approche à composant orienté service vise alors à utiliser l'approche à service en ajoutant des éléments de l'approche à composant [7]. L'approche à composant propose la composition structurelle : une application est un assemblage de composants, cet assemblage est en général décrit grâce à un langage de description d'architecture [8]. Elle propose aussi d'encapsuler la partie métier d'un composant par un conteneur qui prend en charge des besoins génériques non liés à la partie métier. L'approche à composant, seule, ne suffit généralement pas à bénéficier de mécanisme puissant pour gérer le dynamisme : bien que la sélection de composant à l'exécution n'entre pas en contradiction avec l'approche à composant, très peu de modèles à composant le proposent [Escoffier *et al.*, 2007]. De même, la disparition de composants est rarement pris en compte dans les modèles à composants.

Un des buts d'iPOJO est d'extraire et d'automatiser la gestion du dynamisme des services, pour soulager le développeur de service [Escoffier *et al.*, 2007]. Cette gestion du dynamisme est alors assurée par le conteneur du composant, sous forme de *handlers* [9]. Un langage de composition structurelle permet d'exprimer le comportement lié au dynamisme d'un composant iPOJO. Les handlers exploitent ces compositions pour créer ou supprimer des connexions avec d'autres composants. L'aspect service de iPOJO permet aux handlers de découvrir les services disponibles

6. Une interface Java contient un ensemble de signatures de méthodes, mais ne fournit pas d'implémentation. Une interface est implémentée par une classe, qui elle fournit le code exécutable qui réalise les méthodes.
7. Ou inversement, selon le point de vue.
8. ADL en anglais, acronyme de *architecture description language*.
9. « Gestionnaires » en français.

lors de l'exécution, via un annuaire de service. L'environnement d'exécution proposé par iPOJO permet aux composants d'être valides ou non selon l'état des dépendances d'un composant : le cycle de vie des composants est géré automatiquement, le développeur d'un composant n'a donc pas nécessairement à écrire le code qui gère ce dynamisme. Un composant iPOJO peut exhiber des propriétés qui sont configurables, leurs valeurs peuvent être fournies dans la description des instances ou au démarrage du composant, et peuvent être modifiées pendant l'exécution. Enfin, iPOJO est extensible : le mécanisme des handlers est générique, les handlers gérant les dépendances sont fournis avec iPOJO, et il est possible à un développeur d'en définir d'autres. Un composant orienté service avec des handlers est représenté sur la figure 3.4.

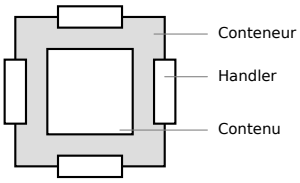


FIGURE 3.4 – Exemple de composant iPOJO ayant quatre handlers. Adaptée de [Escoffier *et al.*, 2007].

Au niveau de la réalisation logicielle, iPOJO se base sur OSGi. Les instances de composants sont exposées sous la forme de services au sein de la plateforme OSGi. Un composant peut dépendre d'autres composants iPOJO, mais aussi de services OSGi. Cette intégration évite de fragmenter les plateformes à services en proposant une autre plateforme non compatible avec l'existant. Un composant iPOJO est une classe Java traditionnelle dans laquelle est injectée les mécanismes de liaisons avec les handlers [10], lors de la compilation en bytecode Java. Des métadonnées viennent compléter cette classe, sous forme d'annotations [11] ou de fichier XML.

3.1.4 Cilia

Cilia est un modèle à composant dédié à la médiation de données. Il a été réalisé dans le cadre d'une recherche doctorale au sein de l'équipe Adèle du laboratoire d'informatique de Grenoble. La thèse a été soutenue en juin 2012 [Garcia, 2012].

La **médiation de données** est un domaine qui vise à l'exploitation de données, sous l'angle de la récupération de données d'une ou plusieurs sources, des traitements des données récupérées, et de l'envoi des données à un ou plusieurs consommateurs. Le but du traitement des données est d'adapter les données fournies pour qu'elles puissent être consommées. D'un point de vue architectural, une médiation de données peut être décomposée comme un ensemble d'entités élémentaires connectées entre elles, ainsi qu'aux fournisseurs et aux consommateurs de données, entités capable d'effectuer les traitements nécessaires pour que les données des fournisseurs puissent être exploitées par les consommateurs. De telles entités sont alors appelées **médiateurs**. Gio Wiederhold définit un médiateur comme « une entité logicielle qui exploite les connaissances sur un certain ensemble ou sous-ensemble de données dans le but de produire de l'information pour une couche applicative de plus haut niveau » [12] [Wiederhold, 1992]. Il énonce cinq tâches

10. D'où le nom iPOJO : injected POJO, POJO signifiant *plain old Java object*, c'est-à-dire un objet Java ordinaire.

11. Les annotations permettent de déclarer très facilement des propriétés aux différents constituants d'une classe Java. Voir `http://docs.oracle.com/javase/6/docs/technotes/guides/language/annotations.html`.

12. Proposition de traduction pour « A mediator is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications ».

qui couvrent l'ensemble des fonctionnalités que peuvent remplir les médiateurs vis-à-vis des données : la sélection, la fusion, la réduction, l'abstraction et la généralisation. De cette définition et des tâches transparaient le changement de signification, mais aussi l'acquisition de sens : la donnée devient information.

Cilia est un modèle à composant qui vise à développer des composants faiblement couplés pour construire des applications de médiation flexibles, qui peuvent être distribuées et dynamiquement changées à l'exécution [Garcia *et al.*, 2010]. Puisqu'ils sont destinés à faire de la médiation, les composants sont appelés médiateurs. Comme le montre la figure 3.5, un médiateur est composé de trois parties : un planificateur, un processeur et un répartiteur [13]. Ces trois parties correspondent à trois rôles attribués aux médiateurs vis-à-vis des données qu'ils ont à traiter. Cette décomposition est également le reflet du traitement des données dans le temps au sein d'un même médiateur : reçue par un médiateur, une donnée – ou un ensemble de données – est dans un premier temps traitée par le planificateur, qui effectue un traitement sur celle-ci, puis la passe éventuellement à un processeur, qui effectue à son tour un traitement, puis la passe éventuellement à un répartiteur, qui de même effectue un traitement et la sort du médiateur.

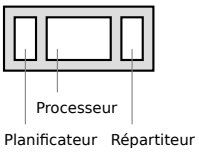


FIGURE 3.5 – Médiateur Cilia. Adaptée de [Garcia *et al.*, 2010].

Premièrement, le planificateur a deux rôles : agrégation de données et déclenchement du traitement du processeur. L'agrégation de données peut être basée sur une corrélation temporelle, pour faire de la synchronisation par exemple, ou sur une corrélation conceptuelle, pour préparer un traitement sur un ensemble de données comme un calcul de moyenne. Deuxièmement, le processeur a pour rôle le traitement métier sur les données : filtrage, opération mathématique, transformation de type, fusion ou fission de données, etc. Enfin, le répartiteur est en charge de choisir la destination des données.

Cilia propose un langage dédié pour la construction de chaines de médiation composées de médiateurs et de connexions. La connexion entre médiateurs se fait avec des liaisons [14]. Une liaison est définie par deux entités, un expéditeur et un collecteur [15], qui implémentent un protocole de communication ; un même expéditeur ou collecteur peut participer à plusieurs liaisons. La gestion de l'hétérogénéité des protocoles de communication est ainsi sortie du médiateur, maximisant d'un côté la réutilisation des planificateurs, processeurs et répartiteurs qui sont écrits pour traiter des données indépendamment des protocoles de communication par lesquels les données arrivent et partent, et maximisant de l'autre côté la réutilisation des expéditeurs et collecteurs qui sont uniquement liés à un protocole de communication et ne contiennent pas de traitement spécifique. Au sein d'une chaine de médiation Cilia, les données sont poussées vers le collecteur suivant par les expéditeurs, et reçues passivement par les collecteurs. À la frontière d'une chaine de médiation, des médiateurs particuliers, appelés adaptateurs, permettent de s'affranchir de se modèle de communication pour être à l'initiative de la récupération de données. Un exemple de chaine est représenté sur la figure 3.6.

Un médiateur est décrit en fonction du type de planificateur, du type de processeur et du type de répartiteur qui le composent. À son tour, une chaine de médiation est décrite en fonction

13. Proposition de traduction pour *scheduler*, *processor* et *dispatcher*.
14. Traduction de *binding*.
15. Traduction de *sender* et de *collector*.

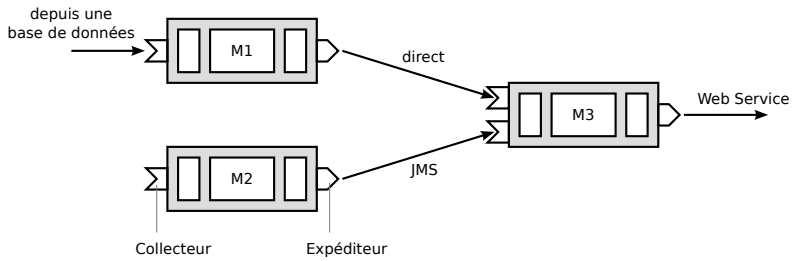


FIGURE 3.6 – Exemple de chaine de médiation Cilia : la chaine fait parvenir des données d'une base de données vers un Web Service. Trois médiateurs participent à cette chaine, le protocole JMS est employé entre deux médiateurs. Adaptée de [Garcia *et al.*, 2010].

des médiateurs qui la composent et des liaisons qu'entretiennent ceux-ci. Ces descriptions sont déclaratives, elles peuvent être écrites dans un fichier que la plateforme Cilia peut exploiter, ou réalisées lors de l'exécution via une suite d'appels à une API exposée par la plateforme. En plus de construire des instances de médiateurs et chaines, cette API permet d'en détruire, et de connaitre toutes les instances en cours d'exécution.

Au niveau de l'implémentation, Cilia repose sur iPOJO, notamment en exploitant son extensibilité via les handlers. Les planificateurs, processeurs et répartiteurs sont implémentées par des classes Java, et peuvent donc exploiter toutes les possibilités du langage.

3.1.5 Plateformes pour des environnements pervasifs

Depuis le milieu des années 1990, et plus particulièrement depuis le début des années 2000, de nombreuses plateformes pour l'informatique pervasive ont été conçues. Déjà en 2005, un article présentait l'analyse d'une trentaine de plateformes [Endres *et al.*, 2005]. Ces plateformes prennent différentes formes : « méta-système d'exploitation », *framework*, modèle d'architecture ou encore modèle de développement. Elles abordent différents aspects de l'informatique pervasive : informatique répartie et partage des informations, mobilité des dispositifs, hétérogénéité, etc. Néanmoins, ces plateformes proposent toutes des mécanismes qui permettent aux applications de s'adapter, notamment au dynamisme des environnements. Ces mécanismes sont variés, et peuvent recouvrir la manière de détecter les évolutions d'un environnement, les types de raisonnements et la planification à mettre en œuvre pour tenir compte des évolutions, ainsi que la manière de réaliser les adaptations [Kakousis *et al.*, 2010].

Dans cette section sont présentés quatre plateformes dédiées aux environnements pervasifs. En conclusion nous soulignons leurs similarités, mais aussi leurs complémentarités.

3.1.5.1 iROS

iROS est une plateforme dédiée aux pièces interactives [Johanson *et al.*, 2002]. « iROS » est l'acronyme de *interactive room operating system*. Comme son nom l'indique, elle vise à faciliter les interactions au sein d'une pièce, entre les utilisateurs d'une part et les dispositifs de sortie et applications d'autre part. Les dispositifs de sorties et les applications sont regroupés sous le terme « service ». Trois tâches à réaliser par iROS ont été identifiées : le déplacement des données entre les interfaces graphiques des utilisateurs et les services, le déplacement des interfaces graphiques de contrôle, et la coordination d'applications dynamiques. iROS est décomposé en

trois sous-systèmes, qui chacun prennent en charge une tâche : le tas de données [16], iCrafter et le tas d'évènements [17]. Le tas de données permet le déplacement des données, iCrafter permet la génération et le déplacement des interfaces de contrôle, et le tas d'évènements permet la coordination des applications dynamiques.

La communication entre les applications se fait via le tas d'évènements, où chaque application poste des évènements. Un évènement est une collection de triplets composés d'un nom, d'un type et d'une valeur. Le tas est en charge d'envoyer aux applications intéressées les évènements qu'il reçoit. Pour cela, les applications déclarent quelles informations les intéressent.

Le tas de données reçoit et stocke les données des applications. Le format et les attributs de ces données ne sont pas figés à un ensemble prédéfini. De plus, un mécanisme de greffons [18] permet de transformer automatiquement les données d'un format à l'autre, suivant les besoins. Par exemple, il est capable de convertir un fichier PowerPoint en JPEG si le dispositif peut seulement afficher des images.

iCrafter est une plateforme à service qui permet d'annoncer et d'appeler des services, ainsi qu'un générateur d'interfaces utilisateur pour les services [Ponnekanti *et al.*, 2001]. Un utilisateur utilise iCrafter pour choisir les services qu'il souhaite contrôler. iCrafter affiche alors une interface. Cette interface est générée à partir d'une description du service. Une description de service contient l'ensemble des opérations du service et leurs paramètres. Ces descriptons reposent sur des interfaces (au sens type abstrait) prédéfinies, par exemple « PowerSwitchInterface », « DataConsumer », « DataProducer », etc. iCrafter est capable d'afficher des interfaces utilisateur qui agrègent des services, s'il existe un générateur capable d'agréger ces services. Par exemple, un générateur peut être capable de créer une interface graphique qui agrège les deux services « DataConsumer », « DataProducer ». Ceci serait le cas d'un utilisateur qui veut utiliser conjointement un appareil photo et une imprimante. La description de service de l'appareil photo implémente l'interface « DataConsumer », la description de service de l'imprimante implémente l'interface « DataProducer ». S'il existe un générateur d'interfaces qui prend en compte ces deux interfaces, alors l'utilisateur aura une interface agrégée qui permet en une seule fois de prendre une photo puis d'imprimer cette photo, plutôt que de passer par un fichier intermédiaire temporaire. Chaque générateur est dédié à un langage d'interface utilisateur. HTML et VoiceXML sont deux exemples de langage d'interface.

3.1.5.2 Gaia

Gaia est une plateforme dédiée aux espaces interactifs [Román *et al.*, 2002]. Elle propose une infrastructure qui coordonne des entités logicielles et des dispositifs hétérogènes en réseau, au sein d'un espace interactif. L'utilisateur peut interagir avec plusieurs dispositifs et applications simultanément. Gaia propose des services qui permettent d'interroger, d'accéder et d'utiliser les ressources existantes. Gaia permet ainsi de développer des applications centrées utilisateurs, multidispositifs, sensibles au contexte, et qui tiennent compte des ressources.

Gaia propose cinq services : le gestionnaire d'évènements, le service de contexte, le service de présence, l'inventaire des ressources et le système de fichier contextuel.

Gestionnaire d'évènements permet aux applications d'être notifiées de changements comme le démarrage d'un composant, la migration d'une application ou l'entrée d'un utilisateur

16. Traduction de *data heap*.
17. Traduction d'*event heap*.
18. Traduction de *plug-in*.

dans l'espace interactif. Le modèle de communication repose sur trois entités : le fournisseur, le consommateur et le canal. Un fournisseur envoie des évènements, un consommateur reçoit des évènements, un canal transmet les évènements produits par des fournisseurs aux consommateurs. Les consommateurs doivent s'enregistrer auprès d'un canal pour recevoir les évènements transmis par ce canal. Gaia même propose quelques canaux qui délivrent les évènements liés aux nouveaux services, applications, personnes, etc. Une application peut définir ses propres canaux pour notifier ses changements.

Service de contexte fournit des informations à propos du contexte. À l'inverse du gestionnaire d'évènement où les consommateurs reçoivent directement les évènements, le service de contexte est passif et les applications doivent faire la demande pour obtenir les informations sur le contexte. Le service de contexte propose par exemple : la température, la localisation des personnes présentes, la météo. Des informations de plus haut niveau peuvent être déduites de ces informations. Par exemple le type d'activité qui a lieu à un instant donné peut être inféré par l'analyse des applications en cours d'exécution, des personnes présentes, etc. Un modèle de règle est proposé : il permet d'évaluer des expressions tel que « Context(number of people, room 2401, >, 4) AND Context(application, Powerpoint, is, running) => Context(social activity, room 2401, is, presentation) ».

Service de présence fournit des informations à propos de la présence des applications, des services, des dispositifs et des personnes. La présence des services et des applications repose sur la métaphore du rythme cardiaque [19] : les services et applications doivent envoyer à intervalle régulier un signal. La réception de ce signal permet d'inférer la présence de l'entité. L'arrêt de la réception du signal déclenche une notification de disparition par le service de présence. La présence des dispositifs et des personnes est inférée par un ensemble de capteurs.

Inventaire des sources recense les ressources présentes. Chaque ressource a une description qui décrit ses capacités. Des exemples de capacités de ressources incluent par exemple un nœud d'exécution, une surface d'affichage et une enceinte. L'inventaire est utilisé lors de l'instanciation des applications, en faisant des requêtes de type « Category == 'Device' and Type == 'Display' ». Cela permet une certaine généricité dans la description des ressources nécessaires à une application.

Système de fichiers contextuel structure hiérarchiquement les données liées à un contexte. Les données sont accédées via des requête de type « /location:/RM2401/situation:/meeting directory », cette requête renvoie les fichiers liés à la salle 2401 lorsque la salle est en situation de réunion. Un utilisateur peut insérer dans la hiérarchie ses propres données.

3.1.5.3 WComp

WComp est un modèle à composant visant la composition de web services [20] [Tigli *et al.*, 2009]. Il est développé par l'équipe Rainbow du laboratoire I3S. Ce modèle à composant facilite la conception d'applications pour les environnements pervasifs en proposant une solution pour la gestion de la mobilité, de l'hétérogénéité, et du dynamisme.

WComp repose sur trois approches : les web services basés sur des évènements, les composants et l'adaptation basée sur des aspects d'assemblage. L'utilisation des web services permet de gérer entre autre l'hétérogénéité des implémentations (à la fois langage de programmation

19. heartbeat en anglais

20. Les web services sont un standard le modèle à services qui semble actuellement le plus utilisé, il est basé sur l'utilisation du protocole HTTP et du langage XML.

et plateforme d'exécution). L'utilisation d'évènements permet d'utiliser le modèle architectural de « publication souscription » [21], qui permet la notification d'une manière très découplée. L'utilisation de composants autorise la composition structurelle. Les compositions structurelles peuvent être modifiées à l'exécution pour adapter une application. Les aspects d'assemblage correspondent à un concept créé pour WComp, qui repose sur les aspects [Kiczales *et al.*, 1997]. Ils permettent d'attacher à un assemblage de composants des règles d'ajout ou de retrait de composants, et de connexion ou déconnexion de ports.

WComp définit un composant par une interface. Cette interface est l'ensemble des ports que propose le composant. Un port correspond soit à une méthode, soit à un évènement. La définition d'un assemblage se fait au sein d'un composant conteneur, qui contient l'ensemble des instances de composants et des liaisons qu'elles entretiennent. Une liaison est définie par un évènement et une méthode. Un composant conteneur possède deux interfaces : une interface fonctionnelle qui exporte les fonctionnalités du composant, c'est-à-dire des évènements et des méthodes, et une interface qui permet de contrôler l'assemblage en ajoutant ou retirant des composants et liaisons. Enfin, une structuration hiérarchique est possible en insérant au sein d'un assemblage un composant mandataire qui représente un autre assemblage. Pour adapter dynamiquement les compositions, des aspects d'assemblage peuvent être déclarés.

La programmation orientée aspect est une manière de suivre le principe de la séparation des préoccupations, en séparant le code relatif à des préoccupations transversales du code métier. Schématiquement, des points d'insertion [22] sont définis à certains endroits d'un code métier, dans lesquels est inséré du code défini ailleurs. L'exemple typique est la journalisation : plutôt que d'ajouter en différents endroits le code nécessaire pour enregistrer certaines informations, le code nécessaire peut être centralisé, et inséré au sein du code métier en spécifiant les points d'insertion. Les aspects d'assemblage s'inspirent de cette idée : l'ensemble des points d'insertion possibles sont les composants et les ports d'un assemblage. Le code inséré est un ensemble de description de reconfiguration architecturale.

Une reconfiguration architecturale est exprimée en fonction de variables libres, à la manière du lambda calcul, qui représentent des composants sur lesquels sont spécifiés des ports. Ces variables libres sont réécrites pour un certain assemblage via des expressions exprimées avec le langage AWK [23], ce qui correspond concrètement à la définition des points d'insertion. Une reconfiguration est exprimée avec un ensemble de sept opérateurs, parmi la condition, la séquence, etc. À l'exécution, les aspects d'assemblage sont transformés en un ensemble de primitives de reconfiguration : ajout et retrait de composants, ajout et retrait de liaisons entre ports. Cette transformation s'effectue en quatre étapes : sélections des aspects qui peuvent s'appliquer à l'assemblage, définition des points d'insertion sur les aspects, compositions des aspects selon un ensemble de règles prédéfinies ce qui peut amener à des fusions d'aspects, et enfin transformation en un ensemble de primitives de reconfiguration.

3.1.5.4 Smart-M3

Smart-M3 est une plateforme pour les environnements pervasifs [Honkola *et al.*, 2010] développée par l'entreprise Nokia. Il propose une architecture et des concepts pour gérer l'hétérogénéité, l'accès à l'information et la localisation des dispositifs dans les environnements pervasifs. La plateforme s'appuie sur les concepts du web sémantique [Berners-Lee *et al.*, 2001], en apportant un aspect dynamique.

Smart-M3 repose sur le modèle architectural du tableau noir [24], qui est une manière de parta-

21. Traduction de *publish-subscribe*.
22. Traduction de *pointcut*.
23. AWK est un langage dédié principalement au traitement de chaines de caractères.
24. Traduction de *blackboard architecture model*.

ger des connaissances entre plusieurs entités : ces entités accèdent à une base de connaissances – le tableau noir – pour ajouter, consulter, retirer et mettre à jour les connaissances. Dans Smart-M3, un tableau noir est appelé un courtier d'informations sémantiques [25], les entités sont des processeurs de connaissances [26]. Un ou plusieurs courtiers stockent l'information d'un environnement pervasif. L'information est stockée sous forme de graphe RDF [27]. L'information peut être exprimée selon des ontologies décrite en OWL [28]. Le protocole de communication entre les processeurs et les courtiers n'est pas imposé et peut être basé sur des technologies de l'approche à service. Les processeurs ne communiquent pas directement entre eux : ils passent systématiquement par les courtiers. Ce faible couplage permet l'apparition et la disparition d'un processeur sans impacter les autres. Un exemple d'utilisation de Smart-M3 est représenté sur la figure 3.7.

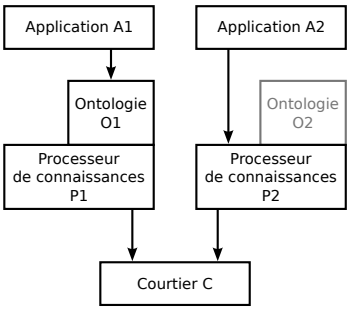


FIGURE 3.7 – Exemple d'utilisation de Smart-M3 : deux applications peuvent communiquer via le courtier, en passant par des processeurs de connaissances. L'application A1 passe par une ontologie O1 pour accéder à l'information, l'application A2 passe directement par le processeur de connaissances. Adaptée de [Honkola *et al.*, 2010].

Comme le montre la figure 3.7, pour exploiter la plateforme Smart-M3, une application utilise un ou plusieurs processeurs pour accéder aux connaissances, ou passe par l'API d'une ontologie. Elle peut alors ajouter, retirer et mettre à jour des connaissances, et bien sûr y accéder au moyen de requêtes, mais aussi s'enregistrer pour être prévenue des changements de connaissances. Pour que les connaissances puissent être exploitables par plusieurs applications, ces applications doivent nécessairement connaitre la sémantique des connaissances. Soit elles les connaissent directement, soit elles passent par des ontologies, pas forcément identiques, mais qui doivent reposer au moins en partie sur les mêmes connaissances. Ainsi, Smart-M3 promeut la standardisation des ontologies propres à un domaine, plutôt que la standardisation des cas d'utilisation d'un domaine.

Une démonstration a été réalisée à partir de cette plateforme, dans lesquels les ontologies sont utilisées pour inférer des objectifs de l'utilisateur [Niezen *et al.*, 2010, van der Vlist *et al.*, 2010]. Cette démonstration permet à un utilisateur de créer des connexions sémantiques entre différents dispositifs. Une interface tangible permet à l'utilisateur de connaitre les connexions possibles entres les différents dispositifs, et de réaliser ces connexions de manière symbolique.

25. Traduction de *semantic information broker*.
26. Traduction de *knowledge processor*.
27. RDF est le sigle de *resource description framework*, un format de description de ressources.
28. OWL, également appelé *Web Ontology Language*, est un langage de représentation de connaissances.

3.1.6 Synthèse

Cette revue des plateformes pour le dynamisme a été scindée en deux parties : les plateformes généralistes et les plateformes dédiées aux environnements pervasifs. Les premières font peu d'hypothèses sur le domaine d'utilisation, et fournissent des mécanismes très génériques qui permettent d'adapter des applications à un environnement dynamique. Elles ciblent les développeurs qui souhaitent bénéficier d'une solution technique pour gérer le dynamisme.

Les secondes sont dédiées à l'informatique pervasive et s'appuient généralement sur des plateformes généralistes. Ainsi, WComp se place au-dessus des services web, Smart-M3 suggère que différentes technologies sous-jacentes peuvent être utilisées, notamment des technologies proposant l'approche à service, ReWiRe – qui n'a pas été présenté ici – s'appuie sur OSGi [Vanderhulst *et al.*, 2008]. Elles exhibent souvent des cas d'utilisations concrets mettant en jeu des dispositifs physiques réels : le dynamisme des dispositifs est traité. La découverte de ces dispositifs, ou plus généralement de services, est un problème identifié : iROS dispose d'un service de découverte, EMI^{2}lets – qui n'a pas été présenté ici – la considère comme une fonctionnalité primordiale [López-de Ipiña *et al.*, 2006], le protocole de communication définit par Smart-M3 inclus les primitives d'arrivée et de départ, etc.

Si quelques plateformes abordent la multimodalité, son traitement n'est pas central. Par exemple, la fusion de données n'est pas abordée. L'accent est mis sur la conception générale d'applications, plutôt que sur la conception d'interactions que peut avoir un utilisateur avec les services. Les plateformes qui concernent plus particulièrement l'adaptation des interfaces graphiques – non traitées ici – abordent ces interactions, mais n'abordent pas l'aspect découverte de dispositifs et récupération des données qui en sont issues.

3.2 Conception d'interfaces multimodales

Pour faciliter la réalisation d'interfaces multimodales, plusieurs outils logiciels issus de la recherche ont été proposés. Ces approches font l'hypothèse de la présence des dispositifs d'interaction. Le bon fonctionnement des interfaces multimodales ainsi créées est alors dépendant de la présence de ces dispositifs. Un dispositif non prévu lors de la conception ne peut pas être pris en compte lors de l'exécution. Ces approches sont utiles si le contexte d'exécution est connu à l'avance. C'est le cas pour la simulation d'un poste de pilotage d'avion [Bouchet *et al.*, 2005], pour les interventions chirurgicales [Mansoux *et al.*, 2007], etc. Ces approches sont également utiles dans le cas de prototypage rapide d'interactions multimodales [Serrano *et al.*, 2008a, Lawson *et al.*, 2009], où le concepteur essaie diverses interactions multimodales avec les dispositifs disponibles dont il dispose.

Dans la suite de cette section, les outils existants pour la multimodalité sont étudiés [29] selon trois volets d'études :

Spécification de l'interface multimodale Nous centrons notre étude des outils existants sur la façon de spécifier l'interaction et la facilité de modification de cette spécification. Pour ce volet nous avons donc deux critères d'analyse :

Modèle de conception Ce critère reflète les concepts proposés à un concepteur. Il vise à évaluer la diversité des solutions proposées par l'ensemble des outils étudiés. La valeur de ce critère est libre, explicitant les concepts centraux que doit manipuler un concepteur ou développeur pour spécifier une interaction.

29. Pour une revue complète des outils pour la multimodalité, nous renvoyons le lecteur à deux thèses, celle de Bruno Dumas [Dumas, 2010] et celle de Marcos Serrano [Serrano, 2010] soutenues en 2010

Adaptation manuelle de l'interface Ce critère reflète la facilité de modification d'une interface par un concepteur ou développeur. Est-ce que le concepteur peut facilement tester différentes alternatives ? À quel point peut-il réutiliser des entités de base ? Peut-il réutiliser des entités plus complexes ? La valeur de ce critère est soit *aucune*, soit *minimale*, soit *correcte*, soit *poussée*.

Intégration de l'interface dans une application interactive Comme expliqué dans le chapitre précédent, nos travaux considèrent une application multimodale avec d'un côté le noyau fonctionnel et de l'autre l'interface multimodale. Aussi il est important dans notre revue des outils existants d'étudier le degré de couplage d'une interface multimodale avec une application. Pour ce volet, nous avons donc un critère :

Intégration de l'interface Ce critère reflète le degré de couplage d'une interface multimodale avec l'application. La valeur de ce critère est soit *interne*, soit *externe*. Interne signifie que le code de l'interface est mêlé au code de l'application ; externe signifie que le code de l'interface n'est pas mêlé au code de l'application. Lorsque l'information n'est pas explicite, la valeur de ce critère est donnée en fonction des exemples fournis.

Dynamisme et adaptation automatique Comme nous consacrons nos travaux aux environnements pervasifs, il convient de noter les supports fournis par les outils existants pour d'une part découvrir dynamiquement des dispositifs d'interaction et d'autre part pour automatiquement adapter l'interaction en conséquence. Deux critères sont retenus ici :

Mécanisme générique de découverte Ce critère reflète d'une part l'effort que doit faire un développeur pour gérer la découverte d'un dispositif, d'autre part la factorisation du code relatif aux protocoles de découverte. Cet effort est faible si le développeur peut s'appuyer sur des mécanismes proposés par l'outil ; il ne l'est pas si le développeur doit lui même écrire le code relatif à la découverte. La valeur de ce critère est soit *oui*, soit *non*.

Adaptation automatique de l'interface Ce critère reflète la capacité d'une interface à réagir à l'apparition ou la disparition d'un dispositif d'interaction lors de l'exécution, ainsi qu'à leurs particularités. Il recouvre plusieurs facettes : une interface multimodale peut-elle être utilisable en l'absence d'un dispositif? Un dispositif peut-il remplacer un autre dispositif équivalent ? L'interface s'adapte-t-elle à la disparition d'un dispositif et à sa réapparition ? La valeur de ce critère est soit *aucune*, soit *minimale*, soit *correcte*, soit *poussée*.

3.2.1 ICon

ICon [30] est un outil réalisé dans le cadre d'une thèse soutenue en 2004 [Dragicevic, 2004]. Ce travail visait à combler le manque d'adaptabilité en entrée des applications. Cette adaptabilité en entrée est définie comme une combinaison de trois propriétés : la contrôlabilité, l'accessibilité et la configurabilité. La contrôlabilité est la capacité d'un système interactif à exploiter efficacement les entrées enrichies. L'accessibilité est la capacité à exploiter des entrées appauvries. La configurabilité est la capacité à pouvoir être finement personnalisé du point de vue des entrées [Dragicevic et Fekete, 2004]. Pour combler ce manque d'adaptabilité en entrée, ICon est un environnement de développement et un environnement d'exécution.

L'environnement de développement utilise les concepts de composants, ports et connexions [31]. Un composant est une boite noire qui interagit avec l'environnement via ses ports. La fonction principale d'un composant est de produire des valeurs de sorties en fonction des valeurs d'entrées.

30. Acronyme de *input configurator*.
31. La terminologie exacte employée est « dispositif », « slots » et « connexion ».

Les ports sont typés. Ils peuvent être reliés entre eux. Deux ports reliés signifient que si une donnée est mise par un composant sur un port de sortie, alors cette donnée pourra être lue sur le port d'entrée relié, par son composant. Les composants peuvent être configurés. Trois types de composants sont définis : les composants systèmes, les composants utilitaires et les composants d'application. Les composants système représentent les ressources du système, notamment les dispositifs. Les composants utilitaires sont des composants génériques, comme des adaptateurs de types et des composants de calcul. Les composants d'application contrôlent l'application.

ICon est prévu comme une configuration d'un logiciel. Le couplage d'une interaction avec l'application est donc très fort. ICon ne se présente pas comme une plateforme indépendante qui permet d'exécuter une interface multimodale, mais comme une extension à ajouter à un logiciel. Ce couplage très fort permet de contrôler une application à plusieurs niveaux. Au niveau de la boite à outil graphique utilisée : un composant peut représenter un objet graphique, comme une barre de défilement. Au niveau de la tâche, un composant peut représenter une fonctionnalité de l'application, comme le tracé de ligne ou de rectangle dans un logiciel de dessin par exemple.

Aucune aide n'est fournie pour la découverte de dispositif, notamment parce qu'ICon repose sur l'exploitation des pilotes de matériel, et ne suppose pas l'accès d'un dispositif par un protocole de découverte. La conception d'une interface se fait au sein de l'environnement où elle est utilisée : parmi les composants systèmes disponibles dans la bibliothèque, ne sont accessibles que ceux qui sont actuellement présents dans la plateforme. À l'exécution, un composant système a un cycle de vie lié au cycle de vie de la configuration à laquelle il appartient : si le dispositif disparait de l'environnement, le composant est maintenu. Donc, l'interface n'est pas adaptée dynamiquement.

L'adaptation est manuelle. Elle repose sur l'existence de composants qui font la transition entre un dispositif et une technique d'interaction, par exemple la manipulation d'un curseur avec le clavier. De même, l'adaptation des types de données est réalisée par des composants dédiés qui doivent être insérés par le concepteur ou développeur. La compatibilité des ports quant aux types de données est représentée graphiquement par l'outil de construction : deux ports envoyant ou acceptant le même type de données sont représentés par la même figure géométrique. Il existe des composants dont les ports s'adaptent aux types de données qu'ils doivent recevoir.

Volet	Critère	Valeur
Spécification de l'interface multimodale	Modèle de conception	composants, ports et liaisons
	Adaptation de l'interface	minimale
Intégration de l'interface dans une application interactive	Intégration de l'interface	interne
Dynamisme et adaptation automatique	Mécanisme générique de découverte	non
	Dynamisme et adaptation automatique	aucune

Table 3.1 – Positionnement de ICon suivant notre grille d'analyse

3.2.2 ICARE

ICARE[32] est un outil réalisé dans le cadre d'une thèse soutenue en 2006 [Bouchet, 2006]. Il vise à faciliter la conception d'interfaces multimodales. Il propose un modèle à composant

32. Acronyme de *interaction-CARE*.

pour la conception d'interfaces multimodales. Deux sortes de composants sont identifiées : les composants élémentaires, les composants de composition [Bouchet *et al.*, 2004]. Les composants élémentaires sont de deux types : les composants de dispositifs d'interaction, et les composants de langage d'interaction. Ce découpage reflète la définition d'une modalité : un couple composé d'un dispositif et d'un langage associé [Nigay et Coutaz, 1997]. Les composants de composition visent à combiner les modalités, combinaisons basées sur les opérateurs CARE [Nigay et Coutaz, 1997]. Trois composants sont alors proposés : composant pour la complémentarité, composant pour la redondance, et composant pour l'équivalence. Enfin, bien qu'ils ne fassent pas partie explicitement du modèle à composant, les composants représentant les tâches élémentaires d'une application sont utilisés pour transmettre les données qu'ils reçoivent vers l'application via un protocole de communication. Un exemple d'utilisation de ces composants est présenté sur la figure 3.8.

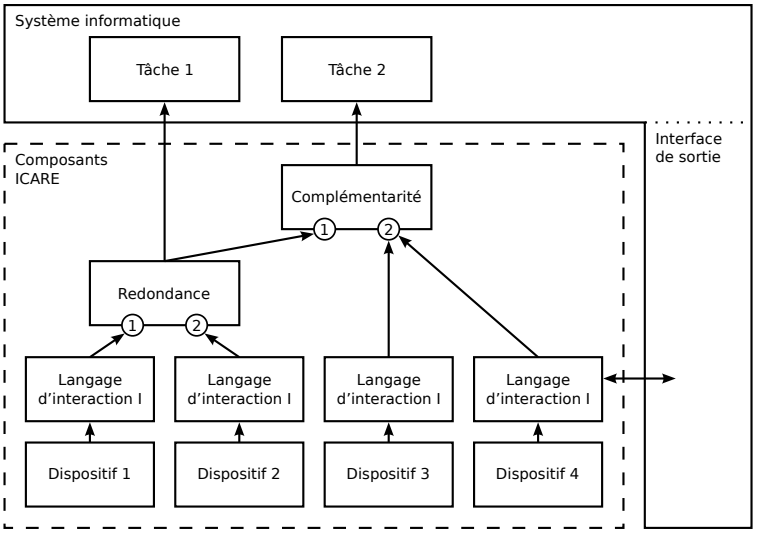


FIGURE 3.8 – Exemple d'utilisation de composants ICARE connectés à un système informatique existant. Adaptée de [Bouchet, 2006, p. 182].

En découplant l'interface multimodale et l'application l'intégration de l'interface multimodale est donc externe à l'application. Les composants de composition sont alors génériques, puisqu'ils ne sont pas liés à une application. De même, ils ne sont pas dépendants des types de données qu'ils reçoivent des composants de langage d'interaction : ils traitent uniquement des évènements, sans s'intéresser aux données particulières que ces évènements encapsulent. Si cette approche est intéressante pour la réutilisation des composants de composition, elle force les composants élémentaires à fournir des données conformes à ce qu'attendent les tâches de l'application.

Le modèle à composant ICARE n'inclut aucun mécanisme pour l'adaptation des interfaces multimodales à l'exécution. Une interface multimodale est entièrement définie lors de la conception. Les composants de dispositifs et les composants de langage d'interaction ont un ensemble de propriétés associées. Ces propriétés sont définies dans le modèle à composant, et une valeur leur est attribuée par les développeurs de composants. Ces propriétés concernent la fréquence d'échantillonnage d'un dispositif, le nombre de degrés de liberté permis, le domaine des valeurs acceptées ou fournies, etc. Elles sont toutes descriptives, dans le sens où elles permettent au concepteur de choisir un dispositif ou un langage d'interaction associé en fonction d'une tâche, mais elles ne sont pas utilisées à l'exécution. L'exception notable est l'exploitation du domaine de valeurs pour autoriser ou empêcher la connexion entre un composant de dispositif et un composant de

langage, lors de l'assemblage de composants.

La modélisation des dispositifs et des langages d'interaction est restrictive : un composant de dispositif ne dispose que d'un port de sortie, un composant de langage ne dispose que d'un port d'entrée et d'un port de sortie. Cette modélisation a le mérite d'être simple, mais elle impose un couplage fort entre un dispositif et un langage d'interaction, limitant fortement la réutilisation d'un composant de langage pour un autre dispositif.

Volet	Critère	Valeur
Spécification de l'interface multimodale	Modèle de conception	composants, ports et liaisons
	Adaptation de l'interface	minimale
Intégration de l'interface dans une application interactive	Intégration de l'interface	externe
Dynamisme et adaptation automatique	Mécanisme générique de découverte	non
	Dynamisme et adaptation automatique	aucune

Table 3.2 – Positionnement de ICARE suivant notre grille d'analyse

3.2.3 Squidy

Squidy est un projet réalisé dans le cadre d'une thèse, soutenue en 2010 [König, 2010]. Squidy permet le développement d'interfaces multimodales, notamment en intégrant des dispositifs d'interaction hétérogènes et diverses boites à outils dédiées à l'interaction [König *et al.*, 2010].

Il se base sur le style architectural *pipe and filter*. Ces deux éléments servent d'intermédiaire entre une source et un puits. Une source est un élément qui génère des données, un puits est un élément qui consomme des données. Un pilote de matériel est modélisé par un puits ou une source. Un filtre est un élément qui reçoit des données sur une entrée, effectue un calcul dessus, et envoie le résultat du calcul sur sa sortie. Un nœud peut être soit une source, soit un puits, soit un filtre. Un tube relie deux nœuds, définissant le chemin des données issues d'un nœud vers un autre nœud. Une canalisation [33] est un ensemble de nœuds lié par des tubes, ainsi qu'une source et d'un puits. Par conséquent, un nœud peut également être une canalisation, fournissant ainsi une structure hiérarchique. Un exemple d'utilisation de tubes et de filtres est représenté sur la figure 3.9.

Les nœuds peuvent être modélisés comme des composants avec au plus un port d'entrée et un port de sortie. Dans ces ports, circulent des ensembles de données. C'est au filtre de trouver les données qui l'intéressent. Pour faciliter cette procédure, une hiérarchie de type de données est fournie. Ces types sont en partie inspirés par les travaux de Wallace [34]. [Wallace, 1976] : ils sont orientés vers l'interaction [35]. Un filtre associe des callbacks (traitant d'évènements) à des types de données, qui sont automatiquement appelées lors de passage de données. Un callback effectue alors les calculs qu'il souhaite. Squidy fournit une bibliothèque de nœuds. Un concepteur les utilise et les connecte entre eux. Les filtres peuvent être configurés, et le code source peut être facilement modifié.

33. Traduction de *pipeline*.
34. Ceux-ci ont été étudiés dans la section 2.1.1.2.1 page 23.
35. Exemples de type : « bouton », « position dans un espace en deux dimension », « geste ».

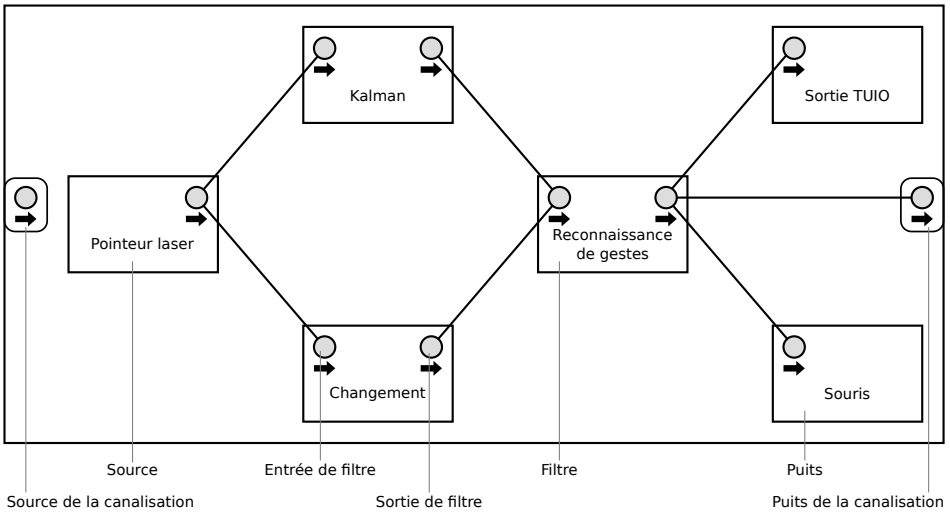


FIGURE 3.9 – Exemple d'utilisation de tubes et filtres Squidy : canalisation mettant en jeu un laser, une souris et une sortie au protocole TUOI (`http://www.tuio.org/`). Adaptée de [König *et al.*, 2010, p. 5].

Un dispositif d'interaction est modélisé par une source ou un puits. Les sources et les puits sont capables d'effectuer les traitements spécifiques pour récupérer des données de l'extérieur, ou en envoyer. Pour gérer les connexions aux dispositifs et les conversions de données entre les types des dispositifs et les types internes à Squidy, un mécanisme de *bridge* est fourni : un *bridge* est propre à un protocole de communication, à une interface native, ou à un dispositif en particulier. Un *bridge* propre à un protocole fournit un très bon moyen d'intégrer des dispositifs hétérogènes. Ceci est cependant à mitiger : convertir une donnée vers un type orienté interaction suppose de connaitre la nature de cette donnée, et pas seulement son type.

Les concepts de puits, de source et de *bridge* placent résolument l'interface multimodale hors de l'application. D'ailleurs, Squidy est plutôt présenté comme un moyen de s'intercaler entre des dispositifs d'interaction que comme un moyen de réaliser une interface multimodale pour une application : en soulignant la flexibilité offerte par ces concepts, les auteurs soulignent son adéquation à des environnements pervasifs.

L'adaptation de l'interface multimodale se fait directement dans l'outil de conception d'interface multimodale proposé par Squidy. Cet outil a plusieurs aspects dédiés au prototypage rapide. Tout d'abord, il propose des modifications profondes très facilement : il permet au concepteur de réaliser des pipelines à partir d'une bibliothèque de nœuds, de configurer les filtres utilisés, et même de modifier le code source, tout cela au sein d'un éditeur graphique avancé incluant des fonctions de zoom. Ensuite, il permet d'exécuter directement les pipelines : l'utilisateur contrôle le cycle de vie de chaque nœud. Enfin, la visualisation des flots de données sous forme graphique en temps réel permet de configurer facilement les filtres. En revanche, Squidy n'a pas de mécanisme d'adaptation automatique, mais il utilise des heuristiques pour présenter au concepteur les composants les plus à même d'être utilisés.

Volet	Critère	Valeur
Spécification de l'interface multimodale	Modèle de conception	tubes et filtres
	Adaptation de l'interface	poussée
Intégration de l'interface dans une application interactive	Intégration de l'interface	externe
Dynamisme et adaptation automatique	Mécanisme générique de découverte	non
	Dynamisme et adaptation automatique	minimale

Table 3.3 – Positionnement de Squidy suivant notre grille d'analyse

3.2.4 OpenInterface

OpenInterface est un modèle à composant issu d'un projet européen éponyme [36]. OpenInterface permet de développer des prototypes d'interfaces multimodales [Lawson *et al.*, 2009] par l'assemblage de composants.

OpenInterface est composé d'une plateforme capable d'exécuter une interface multimodale à partir d'une spécification d'interface. Pour créer une spécification d'interface, un concepteur peut utiliser OIDE ou SKEMMI, qui sont des outils graphiques basés sur l'assemblage de composants dont les ports liés modélisent les chemins des flots de données.

Un composant d'OpenInterface est composé d'un ensemble de facettes. Une facette peut modéliser un pilote de dispositif d'interaction, un algorithme de traitement du signal, la fusion ou la fission de données, des communications réseau ou une interface graphique. Une facette définit des ports [37], un port est soit une source, soit un puits, soit une callback. Une source émet des données, un puits reçoit des données, une callback est une source qui propose un mécanisme d'enregistrement pour les composants intéressés. Un composant est décrit avec un langage de description. Les composants ne peuvent pas être utilisés directement par la plateforme, le concepteur doit générer un mandataire qui représente un composant au sein de la plateforme. Ce mécanisme permet de programmer les composants dans différents langages : Java, C et C++, MATLAB [38] et les langages de .NET.

Une interface multimodale est modélisée par un assemblage de composants interconnectés et configurés. Plusieurs types de connexions sont proposés, ce qui permet de faire gérer des protocoles de communication comme TCP, de la synchronisation, mais aussi de la fusion de haut niveau [Vybornova *et al.*, 2008]. Les connexions entre ports peuvent être faites et défaites lors de l'exécution.

OpenInterface offre deux manières d'initier l'exécution d'une interface multimodale : soit en fournissant un assemblage à interpréter, soit en utilisant une API pour construire l'assemblage au fur et à mesure. Ces assemblages peuvent être conçus avec OIDE ou SKEMMI.

36. Site officiel : `http://www.oi-project.org/`.
37. Traduction de *pin*.
38. MATLAB est un langage dédié aux calculs scientifiques.

3.2.4.1 OIDE

OIDE [39] est un modèle à composant qui se place au-dessus de la plateforme OpenInterface, il vise à concevoir des interfaces multimodales d'une manière flexible [Serrano et Nigay, 2009]. Il reprend en partie le modèle à composant d'OpenInterface. Il s'en distingue par la caractérisation d'un composant selon trois critères : sa généricité, son niveau de spécification et sa position au sein d'un flot de données allant d'un dispositif et à une tâche.

La généricité fait référence à la réutilisation du composant. Un composant est spécifique s'il s'applique à un dispositif ou une application particulière ; il est générique sinon. Le niveau de spécification fait référence à sa capacité à être exécuté : il est un composant logiciel s'il représente une implémentation et peut donc être exécuté, sinon il est une entité d'interaction. Cette caractéristique permet à un concepteur d'interaction de spécifier une interface multimodale à un haut niveau d'abstraction en utilisant des entités d'interaction. Les composants ont alors des caractéristiques orientées interaction comme les attributs physiques pour un dispositif, les attributs mathématiques pour les transformations, etc. Ces composants sont liés directement entre eux, sans passer par des ports. Par la suite, guidé par cette spécification abstraite, un développeur refait un assemblage de composants logiciels qui est, cette fois, exécutable. Dans cette transformation, une entité d'interaction correspond à un ou plusieurs composants logiciels.

Enfin, le flot de données entre un dispositif et une tâche est découpé en quatre parties : la partie dispositif, la partie transformation, la partie composition et la partie tâche. La position d'un composant au sein d'un flot de données correspond à une de ces quatre parties.

Volet	Critère	Valeur
Spécification de l'interface multimodale	Modèle de conception	composants, ports et liaisons
	Adaptation de l'interface	correcte
Intégration de l'interface dans une application interactive	Intégration de l'interface	externe
Dynamisme et adaptation automatique	Mécanisme générique de découverte	non
	Dynamisme et adaptation automatique	aucune

Table 3.4 – Positionnement de OIDE suivant notre grille d'analyse

3.2.4.2 SKEMMI

SKEMMI comme OIDE repose sur le modèle à composant d'OpenInterface [Lawson *et al.*, 2009]. SKEMMI est essentiellement un outil graphique pour le développement d'interfaces multimodales en permettant le développement de composants et leur composition. Il propose une vue d'un assemblage en permettant la liaison de composants et non de ports. À la différence de OIDE, où une entité d'interaction correspond à un ou plusieurs composants logiciels, deux composants reliés constituent seulement une vue où sont masquées les liaisons entre ports. Les composants peuvent être en partie redéfinis en changeant les paramètres et les attributs des ports, mais aussi en supprimant ou ajoutant des ports. Ces redéfinitions peuvent être réalisées après la phase de spécification par un programmeur.

39. Sigle de *OpenInterface interaction development environment.*

Volet	Critère	Valeur
Spécification de l'interface multimodale	Modèle de conception	composants, ports et liaisons
	Adaptation de l'interface	correcte
Intégration de l'interface dans une application interactive	Intégration de l'interface	externe
Dynamisme et adaptation automatique	Mécanisme générique de découverte	non
	Dynamisme et adaptation automatique	aucune

Table 3.5 – Positionnement de SKEMMI suivant notre grille d'analyse

3.2.5 Synthèse

Nous constatons que les outils présentés pour le développement ou le prototypage d'interaction multimodale offrent des concepts similaires, en particulier pour la spécification.

3.2.5.1 Spécification de l'interface multimodale

3.2.5.1.1 Modèle de conception

En termes de spécification de l'interaction, tous proposent un modèle basé sur la métaphore de la liaison : une entité reliée à une autre décrit le cheminement des données qui circule selon ce lien. La description d'une interface multimodale produite par les outils est alors un ensemble d'entités reliées. Elle contient les dispositifs, les entités intermédiaires, les tâches, et les liaisons entre eux. Une liaison entre deux entités repose en général sur la notion de port d'une entité. ICARE l'utilise uniquement pour un type de données, obligeant un fort couplage entre les entités. OIDE le propose selon le niveau de spécialisation : un concepteur relie des entités et un développeur relie des ports. SKEMMI peut les masquer selon la vue utilisée. Squidy n'emploie pas le concept de port : les entités reçoivent un ensemble de données et utilisent celles qui les intéressent.

3.2.5.1.2 Adaptation de l'interface

Les outils font la séparation entre le rôle de concepteur et de programmeur, par l'utilisation de vues différentes, par la proposition de différents types de composants et par la génération de squelette de code. Un concepteur peut adapter manuellement une interface plus ou moins facilement, à l'exécution ou non, mais aucun outil ne considère le rôle de l'utilisateur. ICon uniquement, avec ses possibilités de configuration, considère le rôle de l'utilisateur final, mais ce dernier dispose du même outil que le concepteur : la séparation des deux rôles étant alors peu marquée. Ainsi, à moins de jouer le rôle de concepteur, un utilisateur ne peut pas adapter une interface multimodale à ses besoins.

3.2.5.2 Intégration de l'interface dans une application interactive

Les outils étudiés n'ont pas le même couplage entre l'interface multimodale et l'application. ICon a été conçu pour coupler assez fortement l'interface multimodale, l'interface graphique et l'application. ICARE ne propose pas une distinction explicite entre les tâches d'une application

et les dispositifs. Enfin Squidy, OIDE et SKEMMI découplent clairement l'interface multimodale de l'application.

3.2.5.3 Dynamisme et adaptation automatique

Les outils traitent l'hétérogénéité, mais pas tous au même niveau d'abstraction. ICARE et OIDE gèrent l'hétérogénéité à un haut niveau, en proposant chacun un modèle conceptuel qui permet la description des dispositifs de manière générique. Cependant, ces informations sont principalement destinées au concepteur, afin de lui permettre de sélectionner les dispositifs adéquats pour une certaine tâche. Ces informations ne sont pas utilisées par la plateforme à l'exécution. De plus Squidy et OpenInterface gèrent l'hétérogénéité en permettant l'utilisation de différentes technologies pour coder les entités. OpenInterface va plus loin : utiliser le concept d'adaptateur entre les composants lui permet d'offrir plusieurs motifs d'interaction entre composants.

La description de l'interaction avec les outils existants est principalement statique : un concepteur ne peut pas exprimer de variation. Les adaptations automatiques sont dès lors impossibles : tout est spécifié lors de la conception. Nous notons néanmoins que des outils comme SKEMMI et Squidy permettent de créer ou supprimer des connexions lors de l'exécution. Seul l'outil Squidy permet d'ajouter ou retirer une entité lors de l'exécution mais l'outil ne propose pas d'API ou de module qui permette d'exploiter ce dynamisme en dehors de l'utilisation de l'outil de conception. OpenInterface dispose lui d'une API utilisable lors de l'exécution, mais elle permet seulement la création et la suppression de liens et non des entités.

Les interfaces multimodales obtenues avec les outils ne sont pas adaptées automatiquement lors de l'exécution. L'absence de gestion avancée du dynamisme est assez limitant pour l'adaptation automatique. Ce constat motive nos travaux dont les objectifs et approches possibles sont rappelés dans la conclusion suivante.

3.3 Conclusion

À des fins analytiques, nous avons structuré ce chapitre sur les outils existants en deux parties distinctes, l'une dédiée aux environnements pervasifs et l'autre à la multimodalité. Cette structuration peut amener à penser que la frontière entre les différents types d'outils est franche : ainsi les outils de construction d'interfaces multimodales sont inopérants dans les environnements pervasifs, les plateformes pour les environnements pervasifs ne peuvent ni être utilisées dans un autre contexte, ni être utilisées pour réaliser des interfaces multimodales.

Si c'est le cas pour beaucoup d'outils existants, certains rendent cette frontière plus floue : la composition de services web proposée par WComp et l'adaptation au moyen d'aspects d'assemblage sont des solutions très générales appliquées au domaine de l'informatique pervasive, qui permettent alors de créer des interfaces multimodales ; de plus iROS permet via iCrafter d'utiliser une interface graphique ou vocale ; enfin OpenInterface et Squidy proposent des mécanismes pour gérer le dynamisme.

Cependant, les plateformes pour la construction d'interfaces multimodales ne peuvent pas être utilisées telles quelles dans les environnements pervasifs car la gestion du dynamisme et de l'hétérogénéité des dispositifs n'est pas assez avancée. De même, les plateformes pour les environnements pervasifs n'étant pas dédiés aux interfaces multimodales en entrée, elle ne facilitent pas la spécification précise et adaptée de la fonction d'abstraction multimodale des données issues des dispositifs d'interaction jusqu'aux tâches de l'application.

Partant de ce constat sur les outils existants et vis-à-vis de notre objectif de définir une plateforme logicielle pour la multimodalité en entrée dans les environnements pervasifs, nous identifions deux approches possibles :

- ajouter des mécanismes permettant la gestion de l'hétérogénéité et du dynamisme des dispositifs d'interaction au sein d'un outil de conception d'interfaces multimodales ;
- ajouter des mécanismes permettant la spécification d'interfaces multimodales dans un outil qui fournit des mécanismes pour la gestion de l'hétérogénéité et le dynamisme des services.

Ces deux approches ne sont pas opposées et notre solution conceptuelle que nous décrivons dans le chapitre suivant combine les deux approches : nous partons des outils qui fournissent des mécanismes pour l'hétérogénéité et le dynamisme (approche à composants orientés service, composant de médiation) que nous enrichissons pour pouvoir spécifier finement l'interaction multimodale en entrée en nous reposant sur les modèles de spécification des outils existants pour la multimodalité.

Chapitre 4

Proposition conceptuelle : plateforme pour la multimodalité en environnement pervasif

Le chapitre précédent a souligné les manques actuels de la prise en compte des spécificités des environnements pervasifs dans les outils de conception d'interfaces multimodales. Parallèlement, nous avons mis en évidence des caractéristiques nécessaires aux outils dédiés aux environnements pervasifs pour être bien adaptés au développement d'interfaces multimodales dynamiques. Ce nouveau chapitre présente notre proposition pour combler ces lacunes. Le chapitre suivant explicite cette proposition en présentant la réalisation sous forme d'outils logiciels : DynaMo.

Nous avons concentrés nos effort sur la multimodalité en entrée. Aussi dans la suite de ce mémoire, « interfaces multimodales », « interactions multimodales » et « multimodalité » s'entendent toujours en entrée, de l'utilisateur vers le système.

Notre proposition peut être présentée en trois parties. Après une première section qui présente le principe, une description du cycle de développement est fournie, puis les trois parties mentionnées précédemment sont abordées : un langage de spécification pour les interactions multimodales, une plateforme d'intégration, et enfin un gestionnaire autonomique.

4.1 Architecture conceptuelle

L'objectif est de proposer une solution pour la conception et l'exécution d'interfaces multimodales qui prend notamment en compte deux caractéristiques primordiales des environnements pervasifs : l'hétérogénéité et le dynamisme.

De cet objectif découle une contradiction : un concepteur d'interface multimodale doit pouvoir spécifier le plus précisément possible une interface multimodale, alors qu'il ne peut pas connaitre précisément le contexte d'exécution. Nous proposons donc au concepteur **un langage dédié à la multimodalité** qui permet de spécifier **partiellement** une interface multimodale. Une interface multimodale prend place entre des dispositifs d'interaction et une application. Il peut alors décrire l'interface multimodale d'une application sans faire d'hypothèse sur les dispositifs d'interaction qui seront présents lors de l'exécution. Parallèlement, il peut décrire pour un ou plusieurs dispositifs d'interaction une interface multimodale sans faire d'hypothèse sur les applications qu'ils permettront de contrôler lors de l'exécution. Ce langage est destiné à être utilisé

par un concepteur qui n'a pas forcément de connaissances en programmation informatique. Ainsi, il a été conçu de manière à masquer certains détails techniques.

Lors de l'exécution, l'environnement est constitué d'applications et de dispositifs connus. Nous proposons un **gestionnaire autonomique** capable d'exploiter ces spécifications partielles d'interfaces multimodales, en fonction de l'environnement. Son but est de transformer ces spécifications en une interface multimodale fonctionnelle sous la forme d'une **chaine de médiation**. Il est donc en charge de l'adaptation des interfaces multimodales à un contexte donné. Puisque l'environnement est dynamique, l'interface multimodale doit être adaptée tout au long de son exécution. Le gestionnaire autonomique se base sur des modèles : une description d'interface est un modèle, une interface multimodale fonctionnelle est un modèle. Ainsi, transformer des spécifications en interface fonctionnelle revient à réaliser une succession de transformations d'un ensemble de modèles vers un modèle : le gestionnaire autonomique prend des modèles en entrées et génère un modèle en sortie, comme représenté sur la figure 4.1. Les modèles en entrée sont exprimés par le concepteur, en fonction d'un métamodèle proche du domaine des interfaces multimodales ; ce métamodèle est le langage dédié à la multimodalité. Le modèle en sortie est exprimé par le gestionnaire autonomique, en fonction d'un métamodèle proche du domaine des chaines de médiation. L'utilisateur final peut dans une certaine mesure influer sur le travail du gestionnaire autonomique, pour adapter les interfaces générées à ses préférences.

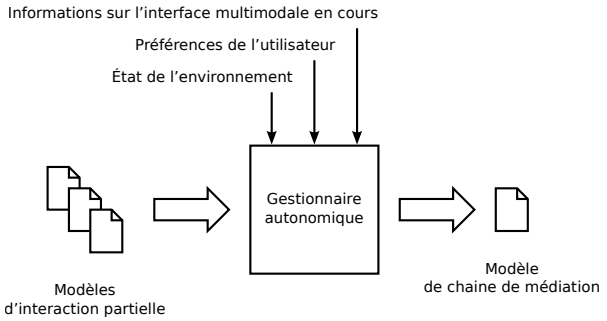


FIGURE 4.1 – Rôle du gestionnaire autonomique : transformation de modèles d'interaction partielle en modèle de chaine de médiation, en fonction de l'état de l'environnement, des préférences de l'utilisateur final et d'informations sur l'interface multimodale en cours.

Pour que le gestionnaire autonomique puisse réaliser son objectif, il doit connaitre l'environnement, et il doit pouvoir faire exécuter les interfaces qu'il génère. Nous proposons une **plateforme d'intégration** composée d'une plateforme à service et d'une plateforme de médiation. La **plateforme à service** permet l'intégration de diverses technologies : elle est capable de découvrir dynamiquement les apparitions et disparitions des dispositifs et des applications, pour notifier le gestionnaire autonomique. Elle est de plus capable de gérer l'hétérogénéité des protocoles de communication pour permettre la communication entre un grand nombre de dispositifs et d'applications. La **plateforme de médiation** permet l'intégration des données : elle est capable de rendre exécutable un modèle de chaine de médiation : création, modification, observation et destruction. La plateforme d'intégration est éminemment dynamique, ce qui lui permet de suivre les évolutions de l'environnement.

Le gestionnaire autonomique, la plateforme de médiation, la plateforme d'intégration et leurs principales interactions sont représentées sur la figure 4.2. Ces éléments constituent notre solution conceptuelle. Nous les détaillons dans la suite de ce chapitre en commençant par exposer le cycle de vie et les acteurs associés.

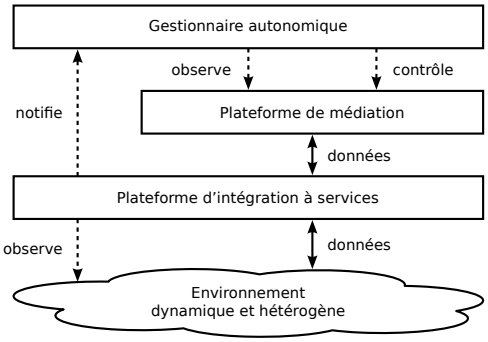


FIGURE 4.2 – Architecture conceptuelle globale.

4.2 Du développement à l'exécution

4.2.1 Trois acteurs

Alors que l'idée de l'informatique pervasive appelle à la disparition des systèmes de la conscience des utilisateurs, la conception de ces systèmes reste une affaire humaine. Ainsi, nous distinguons trois rôles tenus par des acteurs humains lors de la conception et de l'exécution des interfaces multimodales : le développeur, le concepteur d'interactions, et bien sûr l'utilisateur final. Le **développeur** a des connaissances sur la programmation informatique, les formats de données, les protocoles de communication. Le **concepteur d'interactions** n'a pas forcément l'expertise du développeur, mais il possède les connaissances sur les modalités d'interaction et l'utilisabilité et quelques connaissances de base sur l'informatique, notamment qu'un dispositif envoie des données qui peuvent subir des traitements, et qu'une application réagit en fonction des données reçues. Enfin, l'**utilisateur final** n'a qu'une connaissance très partielle du fonctionnement d'une interface, mais il connait les dispositifs et les applications dont il dispose dans son environnement.

4.2.2 Cycle de développement

Le cycle de développement d'une interface multimodale peut se découper en deux étapes : la description de dispositifs et d'applications pour que la plateforme puisse les prendre en compte, suivie de la description d'une interface multimodale. Après ces étapes, une interface multimodale peut être exécutée. La première étape est réalisée par un développeur, la seconde par un concepteur d'interaction tandis que l'exécution est réalisée par une plateforme logicielle.

Dans les approches existantes [1], une interface multimodale peut être exécutée seulement si sa description existe. De plus, une interface multimodale peut être spécifiée seulement si tous les dispositifs et applications qui participent à cette interface ont été décrits : l'interface dépend explicitement des descriptions des dispositifs et des applications. Ce cycle de développement est représenté à gauche sur la figure 4.3. Il repose donc sur la spécification des dispositifs, des applications et de l'interface multimodale.

Dans notre approche, le cycle de développement d'une interaction est plus souple. Nul besoin de décrire explicitement une interface multimodale en choisissant les dépendances en termes de dispositifs et d'application : elle peut être générée à partir des seules descriptions de dispositifs et d'applications. De plus, un concepteur d'interface peut décrire partiellement une interface

1. Présentées au chapitre 3 page 37.

multimodale pour rendre une interface **plus adaptée** à un environnement. Ce cycle de développement est représenté à droite sur la figure 4.3. Le concepteur d'interaction produit des descriptions d'interfaces en amont de l'exécution d'une interface, ces descriptions étant exploitées lors de la génération d'une interface. Une description peut dépendre d'une application seulement, ou bien d'un ou plusieurs dispositifs seulement. L'utilisateur intervient quant à lui après la génération d'une interface particulière, en configurant finement cette interface selon ses préférences. Il peut intervenir également de manière globale sur le processus de génération.

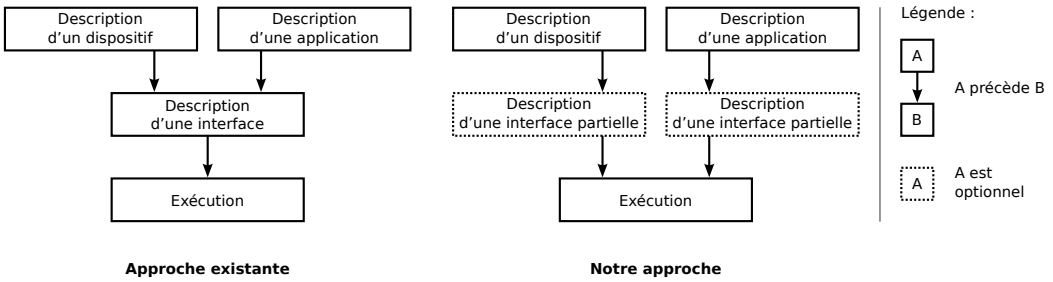


FIGURE 4.3 – Comparaison des cycles de développement : diagramme de précédence des étapes de conception d'interfaces multimodales et de l'étape d'exécution.

4.2.2.1 Description et mandataire d'un dispositif ou d'une application

Ainsi, les seules descriptions des possibilités de dispositifs et d'une application suffit à créer une interface multimodale. Cette caractéristique de notre approche est un moyen de pallier l'absence de description d'interaction. Pour être plus précis, ces descriptions décrivent non pas directement une application ou un dispositif, mais le mandataire [2] d'une application ou d'un dispositif. Un mandataire est une entité logicielle ou code qui s'intercale entre le code d'une interface multimodale et le code d'une application ou d'un dispositif. Comme son nom l'indique, il représente une application ou un dispositif au regard des interfaces multimodales, donnant à ces interfaces une vision **homogène** de l'environnement. Pour qu'un mandataire soit le plus utile possible, il doit être le plus facile et le plus rapide possible à développer par rapport à l'entité qu'il représente. Ce sont les développeurs qui sont en charge de produire ces mandataires et leur description. Parce que ces développeurs n'ont pas forcément de connaissances avancées en interaction, un mandataire propose les mêmes possibilités que l'entité qu'il représente, sans ajouter d'information spécialisée propre aux interactions.

Lors de l'exécution, un mandataire est automatiquement instancié lorsque le dispositif ou l'application qu'il représente est découvert dans l'environnement. Il est supprimé lorsque le dispositif ou l'application disparait de l'environnement. Ces instances sont gérées par la plateforme à services, comme le montre la figure 4.4.

Notre approche propose donc de construire des interfaces multimodales uniquement à partir de descriptions des dispositifs et des applications. Cependant, puisque le processus de génération ne dispose que de peu d'informations, les interfaces générées ne sont généralement pas intuitives pour l'utilisateur ni utilisables. Pour pallier cet inconvénient majeur, des informations peuvent être ajoutées pour guider la génération d'interfaces en spécifiant des descriptions d'interaction.

2. Traduction de *proxy*.

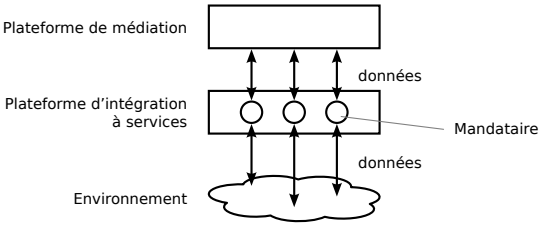


FIGURE 4.4 – Emplacement des mandataires au sein de l'architecture globale.

4.2.2.2 Description d'une interaction

Suite à la création d'un mandataire et de sa description par un développeur, un concepteur d'interaction peut ajouter des informations propres à ce mandataire. Ces informations visent à permettre au processus de génération de créer des interfaces multimodales plus intuitives et utilisables. Comme dans les approches présentées dans le chapitre précédent, il est possible de décrire une interface multimodale directement en fonction d'une certaine application et d'un ou plusieurs dispositifs particuliers. Notre approche autorise donc la spécification complète d'une interface multimodale comme les outils dédiés à l'interaction multimodale [3]. Néanmoins, cette approche n'est pas suffisante pour les environnements pervasifs, car l'environnement d'exécution – c'est-à-dire les applications et les dispositifs accessibles à un instant donné – doit être connu à l'avance. En conséquence, et contrairement aux approches présentées dans le chapitre précédent, une description peut ne pas avoir de dépendances directes vers une application et un ou plusieurs dispositifs, mais vers une application virtuelle et des dispositifs virtuels.

Les notions de **classe de dispositifs** et de **classe d'applications** sont alors introduites. Ces concepts sont à rapprocher des dispositifs virtuels de Wallace [Wallace, 1976] et d'autres auteurs, dans le sens de décrire un dispositif en fonction d'un autre, et des « usages » du protocole USB [usbhid111, 2001], dans le sens d'associer une sémantique à une donnée émise par un dispositif.

La notion de *classe* fait référence à la volonté de réunir des dispositifs similaires et des applications similaires. Une classe de dispositifs est spécifiée par un dispositif virtuel qui est défini par un ensemble de capteurs qui produisent des données, auxquels est attaché un rôle connu par les utilisateurs. Une classe d'applications est spécifiée par une application virtuelle qui est définie par un ensemble de tâches élémentaires consommant des données. Ces tâches ont chacune une sémantique particulière. Les classes ne sont utiles que si elles sont connues par tous les concepteurs d'interaction.

Ainsi, notre approche propose de décrire des interactions qui dépendent d'applications et de dispositifs réels, mais aussi virtuels. L'hypothèse sous-jacente est que de nombreuses applications peuvent être décrites selon quelques classes d'applications, et que de nombreux dispositifs peuvent être décrits selon quelques classes de dispositifs. Si cette hypothèse est vérifiée, l'explosion combinatoire du nombre de descriptions d'interaction nécessaires est considérablement diminuée.

Un calcul naïf du nombre de descriptions d'interfaces unimodales nécessaires pour A applications, D dispositifs, A_v applications virtuelles et D_v dispositifs virtuels donne $A \times D$ descriptions pour l'approche classique, $A \times (A_v + D_v) + D \times (A_v + D_v)$ descriptions avec notre approche. Concrètement, avec 100 000 applications [4], 1 000 dispositifs, 20 applications virtuelles et 20 dispositifs

3. Voir le chapitre 3 page 37.
4. En mai 2012, plus de 450 000 applications pour Android sont disponibles dans la boutique en ligne Google Play. Voir `https://play.google.com/about/features/`.

virtuels : 100 millions de descriptions sont nécessaires avec l'approche classique, « seulement » un peu plus de 4 millions avec notre approche. En considérant qu'une application est décrite uniquement en fonction de l'application virtuelle la plus proche, et qu'un dispositif est décrit uniquement en fonction du dispositif le plus proche, et que chaque application virtuelle est décrite en fonction de tous les dispositifs virtuels, le nombre de description nécessaire avec notre approche baisse à 101 400.

Enfin, la configuration d'une interface peut intervenir en dernier lieu, ce qui permet à l'utilisateur final d'introduire ses préférences personnelles.

4.2.2.3 Configuration de l'utilisateur

Une fois qu'une interface multimodale est générée, l'utilisateur peut la configurer plus finement pour ses besoins propres. Cette étape finale d'adaptation de l'interface par l'utilisateur à ses besoins n'est pas obligatoire. Il existe souvent plusieurs manières d'utiliser un service, bien qu'une description d'interaction réalisée par un concepteur d'interaction tende à améliorer globalement une interface par rapport à une interface générée automatiquement, il est illusoire de penser qu'une certaine interface conviendra à tous les utilisateurs.

Un utilisateur peut donc guider le processus de génération d'une interface multimodale en configurant sa manière préférée d'utiliser une application, et en modifiant les relations entre les capteurs des dispositifs d'interaction et les tâches d'une interface générée.

4.3 Langage de spécification pour les interactions multimodales

Le concepteur d'interaction décrit des modalités d'interaction entre des applications, des dispositifs, des classes d'applications et des classes de dispositifs. Pour cela, il a à sa disposition un ensemble de composants configurables et un moyen d'assembler ces composants entre eux. Un composant encapsule un algorithme qui agit sur les données entrantes et calcule les données qu'il doit émettre. Cet algorithme réalise une fonction qui est décrite dans une documentation accessible au concepteur.

L'approche adoptée s'assimile aux solutions existantes pour la multimodalité présentée au chapitre précédent : la fonction de traitement des données issues des dispositifs jusqu'aux tâches de l'application est décrite par un assemblage de composants. Parmi ces composants, certains comme ceux de fusion implémentent des traitements génériques à la multimodalité. Ces traitements génériques sont proposés au concepteur par des composants configurables.

4.3.1 Composant

Un composant, plus ou moins générique, est développé par un développeur. Ces composants sont à la disposition du concepteur d'interaction pour la conception d'interfaces multimodales. Lors de l'exécution, ils sont accessibles à la plateforme pour permettre l'exécution des interfaces multimodales.

Un composant comprend un algorithme et d'un ensemble de ports. Les ports constituent l'interface du composant à l'extérieur. Deux types de ports sont définis : les ports d'entrée et les ports de sortie. Un port d'entrée peut recevoir des données de l'extérieur et transmettre immédiatement ces données au composant auquel il appartient. Un port de sortie peut recevoir des données de l'intérieur du composant et émettre immédiatement ces données vers l'extérieur. Un composant est schématisé à la figure 4.5.

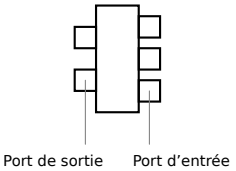


FIGURE 4.5 – Exemple de composant possédant trois ports d'entrée (à droite) et deux ports de sortie (à gauche).

L'algorithme implémente une fonction qui est connue par le concepteur d'interaction. Cet algorithme s'exprime en fonction des données qui proviennent des ports d'entrée, des données qu'il peut émettre sur les ports de sortie, et de variables qui prennent une valeur lors du démarrage du composant. Ces dernières variables permettent aux composants d'être configurables : un algorithme peut être en partie générique, l'application de valeurs aux variables spécialise une instance de composant. L'ensemble des couples composés d'une variable et de la valeur qu'elle doit prendre est appelée une configuration. Pour tout composant, une configuration par défaut est fournie. Un concepteur d'interaction peut modifier cette configuration par défaut, selon les besoins. La configuration est appliquée à un composant lors de son instanciation à l'exécution.

Une instance de composant est un composant qui a une configuration, et qui lors de l'exécution possède un état. La relation entre un composant et une instance de ce composant est la même que la relation entre une classe et un objet de cette classe. Lorsqu'un concepteur d'interaction souhaite utiliser un composant, il déclare une instance et modifie éventuellement sa configuration par défaut. À l'exécution, lorsque la plateforme a besoin d'un composant, elle crée une instance, spécifie une configuration et démarre l'instance.

Ces composants sont destinés à être utilisés au sein d'assemblages qui spécifient l'interaction multimodale.

4.3.2 Assemblage

Un assemblage est constitué de trois ensembles :
- un ensemble d'instances de composants ;
- un ensemble de dépendances ;
- un ensemble de liaisons.

Créer un assemblage se fait de manière déclarative : le concepteur d'interactions déclare les composants qu'il souhaite utiliser, déclare les dépendances de cet assemblage, et définit les liaisons en déclarant les ports qui la définissent. Deux assemblages sont représentés sur la figure 4.6.

L'ensemble des dépendances d'un assemblage constitue les conditions de sa validité. Une dépendance peut être un mandataire de dispositif ou d'application, ou un autre assemblage comme sur la figure 4.6. À l'exécution, lorsqu'un mandataire est automatiquement créé, il devient une dépendance valide, déclenchant en cascade la validité des assemblages ayant une dépendance sur lui. Une dépendance fournit un ensemble de ports qui peuvent être référencés dans l'assemblage.

L'ensemble des liaisons constitue les chemins que les données doivent emprunter. Une liaison est définie par un port d'entrée et un port de sortie : si deux ports sont reliés, alors toute donnée qui transite sur le port de sortie transite sur le port d'entrée. Un port d'entrée – comme un port de sortie – peut participer à plusieurs liaisons. Au sein d'un assemblage, une liaison peut être exprimée en fonction des ports des instances de composants déclarées, et des ports des instances de composants déclarées dans les dépendances.

À partir de cette définition des assemblages, le mandataire d'une application est modélisé comme un assemblage constitué d'un seul composant dont les ports d'entrée représentent les

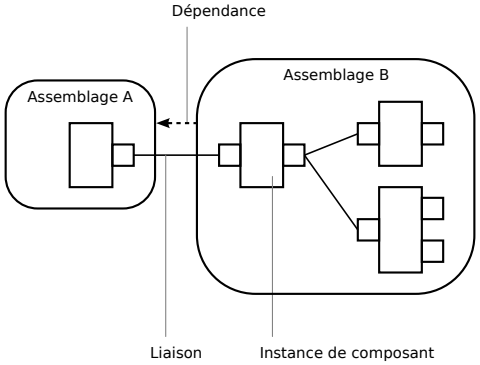


FIGURE 4.6 – Exemple de deux assemblages : l'assemblage B dépend de l'assemblage A, il est composé de trois instances de composants et de trois liaisons dont l'une référence un port de l'assemblage A.

tâches élémentaires et les ports de sortie représentent les différentes informations qu'elle peut émettre. Cet assemblage ne possède ni dépendance ni liaison. Le mandataire d'un dispositif est modélisé de la même manière, à la différence que les ports de sortie représentent ses capteurs et les ports d'entrée représentent ses effecteurs. Ainsi, la définition d'une dépendance peut être simplifiée.

Les notions de dispositifs et d'applications virtuels [5] sont décrites ainsi : un dispositif virtuel – ou une application virtuelle – est un assemblage prédéfini dont la sémantique précise des ports qu'il contient est partagée par les concepteurs d'interaction [6].

Dans notre approche, créer des assemblages est la manière de décrire une interface multimodale. Les dépendances de l'assemblage fournissent l'information sur la validité de l'assemblage. Décrire une interface multimodale consiste à définir les chemins de données entre des dispositifs et un service, ainsi que des transformations à effectuer le long de ces chemins de données.

La création d'assemblage est une activité qui se déroule à l'étape de conception. Les deux sections suivantes concernent la phase d'exécution. La première (section 4.4) porte sur l'exécution même d'une interface multimodale, la seconde (section 4.5) décrit la manière de faire la jonction entre les descriptions effectuées à la conception et leur utilisation durant l'exécution.

4.4 Plateforme d'intégration

La plateforme d'intégration est un ensemble de logiciels qui fonctionnent ensemble et qui ont deux objectifs principaux : gérer dynamiquement les dispositifs et applications présents dans l'environnement, et permettre la communication entre ces différents éléments. Le terme « intégration » fait référence à l'hétérogénéité de l'environnement qui est rendu homogène par l'utilisation de mandataires sur la plateforme, et par la possibilité de créer des canaux de communication évolués entre ces mandataires.

Cette plateforme peut être scindée en deux plateformes distinctes : une plateforme à services qui gère l'hétérogénéité des technologies de découverte et de communication des dispositifs et applications, et une plateforme de médiation qui est responsable plus particulièrement du routage et du traitement des données. Cette distinction peut être approximativement assimilée à la

5. Voir la section 4.2.2.2, page 64.
6. Exemple concret de dispositif virtuel : manette de jeu comme représentée sur la figure 4.16 page 83. Exemple concret d'application virtuelle : lecteur multimédias, navigation sur une carte en deux dimensions.

distinction entre syntaxe et sémantique.

Dans la suite de cette section, nous détaillons la plateforme à services, puis la plateforme de médiation.

4.4.1 Plateforme à services

Tout dispositif ou application apparait comme un service dans la plateforme, même si leur technologie n'est pas basée sur l'approche à services. La création de mandataires est possible par un module chargé de la découverte de ces dispositifs et applications. Un mandataire est un service tel que défini dans l'approche à service. Ce module de découverte est composé de sous-modules, chaque sous-module étant responsable d'un protocole de découverte particulier, par exemple Bluetooth, D-Bus, Web Service, etc. Contrairement aux approches présentées précédemment [7], où chaque dispositif et chaque application dispose de son propre module de découverte, cette approche permet un passage à l'échelle. Imaginons que plusieurs milliers de dispositifs et d'applications puissent être reconnus par un système de création d'interfaces multimodales. Si la découverte est faite par dispositif, alors plusieurs milliers de modules de découverte fonctionnent en même temps. Si au contraire, la découverte est faite par protocole de découverte, et que ces milliers d'applications et dispositifs utilisent une dizaine de technologies différentes, alors seule une dizaine de modules de découverte fonctionnent en même temps.

Découvrir un dispositif ou une application ne passe pas par le démarrage d'un module particulier, mais par la configuration d'un service de découverte propre au protocole de découverte. Dans chaque protocole de découverte, il existe une manière de différencier les entités à découvrir. Par exemple, Bluetooth repose sur l'utilisation d'une chaine de caractères qui caractérise un dispositif : le *friendly name* [8]. Ainsi, la configuration d'un module de découverte se fait en général par l'ajout au module d'une chaine à reconnaitre.

Une fois que le module de découverte a reconnu une application ou un dispositif, il instancie un mandataire qui représente cette application ou ce dispositif au sein de la plateforme. Ce mandataire étant un service, cette instanciation réalise le passage d'une technologie qui n'est pas basée sur l'approche à services vers une approche basée sur les services. Ce mandataire est capable de communiquer avec l'application ou le dispositif avec le protocole de communication idoine. Étant un service, le mandataire s'inscrit dans le registre propre à l'approche orientée services : il est alors disponible au sein de la plateforme d'intégration. Communiquer avec le mandataire revient à communiquer avec l'application ou le dispositif qu'il représente. La disparition de l'application ou du dispositif déclenche le processus inverse : le module de découverte détruit le mandataire, qui déclenche sa disparition du registre. Il convient de noter que certains protocoles de découverte permettent la découverte de l'arrivée d'un dispositif, mais pas sa disparition. Dans ce cas, c'est le mandataire lui-même qui est notifié de cette disparition, par la coupure de la connexion avec l'entité qu'il représente par exemple. Le mandataire doit alors prévenir lui même le sous-module de découverte concerné par ce protocole. L'organisation de la découverte est résumée sur la figure 4.7.

La découverte se fait donc en deux temps. Premièrement, le module de découverte sonde l'environnement dans le but d'instancier des mandataires ou de détruire des mandataires précédemment instanciés. Deuxièmement, l'instanciation ou la disparition d'un mandataire est répercutée dans le registre de la plateforme. Entre les moments où un mandataire est créé et détruit, il peut participer à une interface multimodale. La section suivante décrit de quelle manière.

7. Voir la section 3.2 page 49.

8. Le terme « friendly name » fait référence à un nom compréhensible par un utilisateur, plutôt que par une désignation technique. Voir la section 2.1.1.2.2 page 26 qui présente Bluetooth.

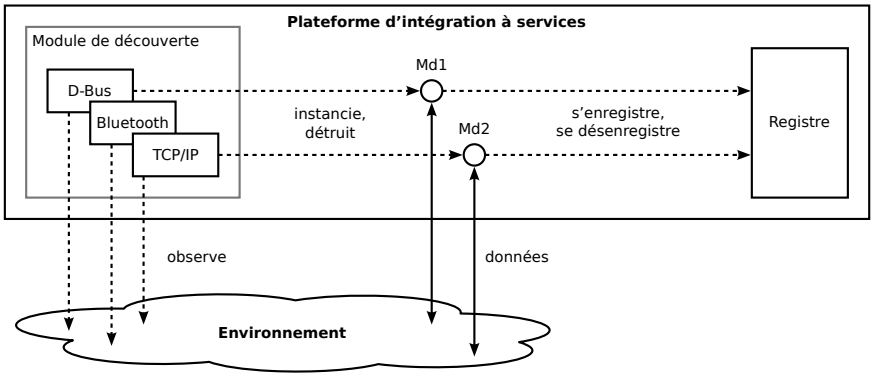


FIGURE 4.7 – Organisation de la découverte au sein de la plateforme à services : trois sous-modules de découverte observent l'environnement, deux mandataires *Md1* et *Md2* ont été instanciés. Ils se sont enregistrés dans le registre et échangent des données avec les entités qu'ils représentent. Par exemple, le sous-module D-Bus a découvert une application en observant l'environnement, il a alors instancié *Md1*, qui s'est enregistré dans le registre, et qui est capable de communiquer directement avec l'application. Lorsque le sous-module D-Bus détectera la disparition de cette application, il détruira *Md1* qui, juste avant d'être détruit, se désenregistrera du registre.

4.4.2 Plateforme de médiation

Au sein de la plateforme d'intégration, des mandataires sont créés et détruits au gré de l'évolution de l'environnement. Pour qu'une interface multimodale soit utilisable, les données qui transitent par les mandataires des dispositifs doivent cheminer jusqu'au mandataire d'une application. L'organisation de ce cheminement peut être vue comme de la médiation de données, c'est-à-dire comme un routage et un ensemble de transformations des données produites à un endroit donné vers un endroit où elles sont consommées. C'est pourquoi une organisation de chemins de données entre des médiateurs est appelée dans notre approche une **chaine de médiation**. Pour un environnement donné, les mandataires sont créés par le module de découverte, puis une chaine de médiation est créée pour acheminer et traiter des données entres ces mandataires.

Pour isoler cette gestion des chaines de médiation, une plateforme de médiation a été définie, elle est représentée sur la figure 4.2 page 62. Ainsi, la plateforme à services gère l'intégration des technologies nécessaires aux interfaces multimodales, ce qui permet à la plateforme de médiation de bénéficier d'une vue homogène de l'environnement et de gérer plus particulièrement l'intégration des données entre des dispositifs et une application. Un exemple de chaine de médiation est représenté sur la figure 4.8. Entre les mandataires et les médiateurs circulent des données en provenance et à destination des dispositifs et applications.

La plateforme de médiation fournit un socle pour la création, l'évolution et l'observation de ces chaines de médiation. D'un point de vue fonctionnel, elle prend en entrée un modèle de chaine de médiation, et exécute une chaine de médiation conforme à ce modèle. Elle permet d'observer le fonctionnement d'une chaine de médiation en cours d'exécution en remontant des informations. Ces caractéristiques sont représentées sur la figure 4.9. La plateforme de médiation est générique ; si elle permet l'exécution d'interface multimodales, c'est parce que les modèles qui lui sont fournis représentent des interfaces multimodales et que les extrémités des chaines sont des mandataires de dispositifs et d'applications.

La section suivante décrit comment sont générées ces chaines de médiation, c'est-à-dire la

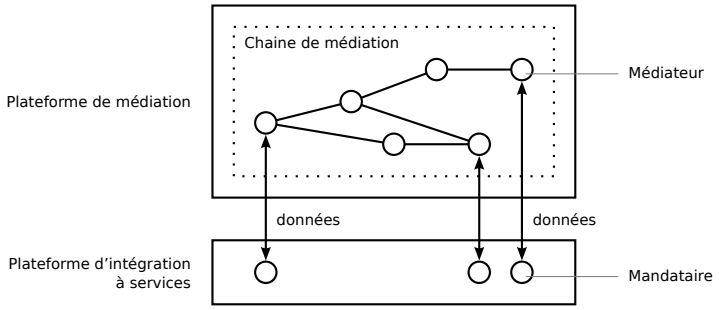


FIGURE 4.8 – Exemple de chaine de médiation : elle est constituée de six médiateurs, et est en rapport avec trois mandataires.

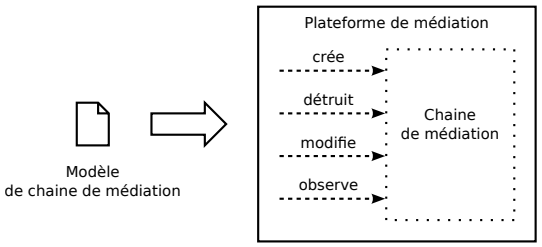


FIGURE 4.9 – Rôle de la plateforme de médiation : elle crée et maintient une chaine de médiation conforme à un modèle qui lui est fourni.

manière de produire le modèle de chaine de médiation qui permet l'exécution d'une interface multimodale.

4.5 Gestionnaire autonomique

Pour faire la liaison entre les descriptions d'interfaces produites par un concepteur d'interaction, les préférences de l'utilisateur et l'état d'un environnement, un module appelé « gestionnaire autonomique » est défini. Le terme « autonomique » fait référence à l'adaptation automatique des interfaces multimodales en fonction du contexte d'exécution et la prise en compte des préférences de l'utilisateur. L'objectif de ce gestionnaire est d'exploiter la plateforme d'intégration décrite dans la section précédente pour créer une interface multimodale en utilisant toutes les informations disponibles.

Le gestionnaire autonomique connait l'état de l'environnement en utilisant le registre de la plateforme d'intégration, qui référence tous les mandataires. Il connait les préférences de l'utilisateur en interrogeant un module dédié à ces préférences. Il a à sa disposition des dépôts contenant les descriptions des applications et dispositifs, ainsi que les descriptions d'interaction disponibles.

Le gestionnaire autonomique a été conçu avec l'hypothèse que seule une application est présente dans l'environnement : l'utilisateur final choisit parmi les applications découvertes celle avec laquelle il souhaite interagir. Ainsi, le registre ne permet pas au gestionnaire autonomique de connaitre toutes les applications disponibles et ne lui permet de voir que l'application choisie par l'utilisateur. Il s'agit ici d'une simplification : dans une deuxième étape, un module chargé d'interpréter un environnement pourrait sélectionner automatiquement cette application en fonc-

tion de la tâche que souhaite faire l'utilisateur. La limitation à une seule application n'est pas majeure : d'une part seules quelques parties du gestionnaire devraient être mises à jour pour tenir compte de plusieurs applications, d'autre part il n'y a pas d'incompatibilité structurelle dans la conception actuelle du gestionnaire. De plus, il est possible de gérer plusieurs applications sans modifier le gestionnaire autonomique : il suffit de générer plusieurs interfaces – une par application – et intercaler un module entre les mandataires de dispositifs et les interfaces générées qui dirigent les données vers telle ou telle interface en fonction de ce que souhaite faire l'utilisateur.

4.5.1 Principe de fonctionnement

Le travail du gestionnaire autonomique est central. C'est en effet lui qui est responsable de l'adaptation des interfaces multimodales en fonction des informations disponibles à un instant donné. Ce qui a été appelé « descriptions d'interaction » jusqu'ici sont en fait des modèles. La plateforme de médiation est capable d'instancier une chaine de médiation conforme au modèle qui lui est fourni. Le travail du gestionnaire autonomique est alors basé sur les modèles : il transforme des modèles spécifiques à la multimodalité, donc compréhensibles par un spécialiste en interaction, vers un modèle d'une chaine de médiation, compréhensible par la plateforme de médiation [9].

Son fonctionnement se décompose en deux étapes principales. La première étape consiste à sélectionner des modèles d'interaction parmi tous les modèles disponibles. La seconde étape consiste à manipuler les modèles sélectionnés pour n'obtenir plus qu'un modèle de chaine de médiation. Cette manipulation consiste à transformer et composer les modèles entre eux, déduire les médiateurs nécessaires, leur fournir une configuration et lier les ports disponibles entre eux. Ces deux étapes sont représentées sur la figure 4.10. Le gestionnaire autonomique est également capable de pallier l'absence de modèles d'interaction, c'est-à-dire de construire une interface multimodale en se basant uniquement sur les modèles de mandataires.

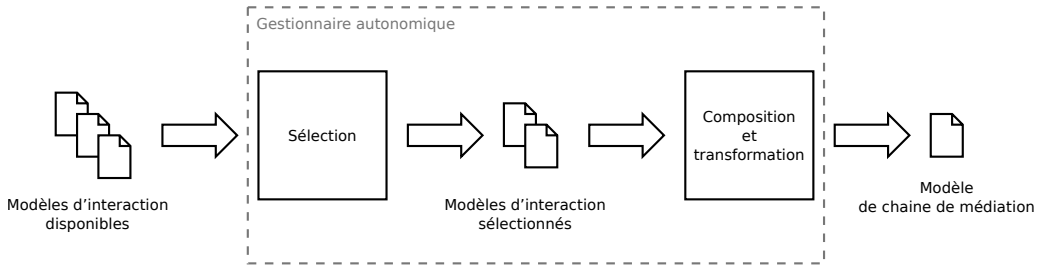


FIGURE 4.10 – Détail du fonctionnement du gestionnaire autonomique : sélection puis composition et transformation.

Les deux sections suivantes abordent d'abord la sélection de modèles, puis la composition des modèles sélectionnés.

4.5.2 Sélection de modèles

Les dépôts des modèles, c'est-à-dire les endroits physiques où sont stockés les modèles d'interaction, sont accessibles par le gestionnaire autonomique. La sélection est nécessaire parce que, d'une part, le nombre de modèles est potentiellement grand et que, d'autre part, ils ne sont pas forcément utiles en fonction de l'état de l'environnement. Enfin, des modèles peuvent être

9. Voir la figure 4.1 page 61.

incompatibles entre eux. La sélection de modèles se fait en deux temps. Premièrement, seuls les modèles utiles sont retenus. Deuxièmement, des heuristiques sont utilisées pour sélectionner les modèles les plus à même de construire une interface multimodale adaptée à l'environnement et à l'utilisateur final.

Comme les interfaces sont décrites partiellement, les sélections s'effectuent non pas sur les modèles même, mais sur des ensembles de modèles cohérents. Il est en effet plus intéressant de comparer des interfaces complètes plutôt que des fragments d'interfaces. Un ensemble est cohérent si ses modèles permettent d'obtenir une interface fonctionnelle après être composés. Ainsi, ce sont des ensembles de modèles cohérents qui sont sélectionnées. Un modèle sélectionné est alors un modèle qui appartient à au moins un ensemble sélectionné. La finalité du processus de sélection est d'obtenir un unique ensemble de modèles. Ces modèles peuvent alors être transformés et composés pour donner une interface multimodale exécutable.

Les dispositifs et les applications peuvent être vus comme des modèles [10]. L'ensemble des modèles disponibles peut être assimilé à un graphe orienté ou non dans lequel un sommet correspond à un modèle et un arc $(m1, m2)$ correspond à une dépendance de $m1$ vers $m2$. Le graphe orienté formé des modèles disponibles ne doit contenir aucun circuit.

4.5.2.1 Propriétés d'un ensemble de modèles

À partir des modèles disponibles, il est possible de générer énormément d'ensembles de modèles. Cependant, chaque ensemble ne produit pas forcément une interface utile et utilisable. Plusieurs propriétés sont définies sur les ensembles de modèles : un ensemble peut être valide, complet, minimal, cohérent. De plus, un ensemble de modèles peut contenir des modèles incompatibles. L'objectif de ces propriétés est de disposer de moyens **intuitifs et calculables** de vérifier qu'un ensemble de modèles peut produire une interface exécutable utile et utilisable. Les définitions qui suivent s'appliquent à des ensembles de modèles. Comme une dépendance définit une condition de validité d'un modèle [11], si un modèle appartient à un ensemble, alors ses dépendances doivent également appartenir à cet ensemble.

4.5.2.1.1 Validité

L'ensemble des modèles disponibles peut être conséquent. Parmi ces modèles, comme énoncé précédemment, certains ne sont pas utiles selon les dispositifs et applications présents dans l'environnement. Par exemple, un fragment d'interface de Wiimote est inutile si la Wiimote n'est pas présente dans l'environnement : ce fragment est invalide au regard de l'état de l'environnement. La propriété de validité d'un ensemble est alors définie en fonction d'un état d'environnement. **Un ensemble de modèles est valide s'il ne contient aucun modèle de dispositif ou d'application absent de l'environnement**.

4.5.2.1.2 Complétude

Les modèles représentent des fragments d'interfaces. L'objectif est néanmoins d'obtenir une interface utilisateur qui permette de relier des dispositifs à des applications, c'est-à-dire d'obtenir une interface en entrée complète. Ainsi, **un ensemble de modèles est complet s'il contient au moins un modèle de dispositif et au moins un modèle d'application**. Les ensembles valides, complets et minimaux ont une propriété intéressante : si une seule application est présente dans l'environnement, alors tout ensemble de modèles valide et complet peut être assimilé à un graphe connexe.

10. Voir la section 4.3.2 page 66.
11. Voir la section 4.3.2 page 66.

4.5.2.1.3 Compatibilité

Parmi les modèles disponibles, il peut exister des modèles en concurrence, c'est-à-dire qu'ils peuvent être utilisés les uns à la place des autres mais pas ensemble : ils sont incompatibles entre eux. **Deux modèles sont incompatibles s'ils sont susceptibles d'exploiter une même action de l'utilisateur final** [12]. Rapporté à un dispositif, deux modèles sont incompatibles par rapport à un dispositif s'ils utilisent un même port de ce dispositif. En effet, si deux modèles utilisent un même port, alors les données issues de ce port pourront être utilisées de deux manières différentes dans l'interface multimodale. Un exemple de modèles incompatibles est donné par la figure 4.11.

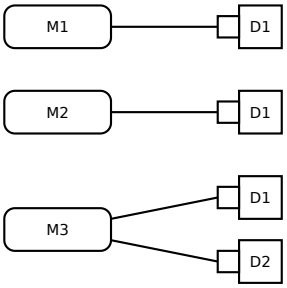


FIGURE 4.11 – Exemple de modèles d'interaction incompatibles : M1, M2, et M3 sont incompatibles car ils utilisent tous l'unique port du dispositif D1. S'ils sont utilisés ensembles, et qu'une données est émise sur le port, alors chaque fragment d'interface interprètera cette donnée à sa manière, en ignorant qu'elle est aussi utilisée par deux autres fragments d'interface.

La notion de compatibilité par rapport à un dispositif peut être généralisée à un ensemble de modèles. Au sein d'un ensemble de modèles, il existe des modèles incompatibles si au moins une des deux conditions est vérifiée :

- il existe au moins deux modèles qui sont incompatibles par rapport à un dispositif ;
- il existe un cycle au sein du graphe non-orienté formé de cet ensemble de modèles ôté des modèles de dispositifs.

Ainsi, si une seule application est présente dans l'environnement, alors tout ensemble de modèles valide, complet, minimal, sans modèle incompatible et ôté des modèles de dispositif peut être assimilé à un arbre [13].

4.5.2.1.4 Minimalité

Deux ensembles de modèles différents peuvent produire une interface équivalente. Par exemple, si un ensemble de modèles produit une certaine interface, ajouter des modèles qui n'ont pas d'influence produira un autre ensemble sans changer les possibilités de l'interface : ce nouvel ensemble de modèle n'est pas minimal. C'est le cas des modèles sans dépendance. Cette possibilité complique les heuristiques de sélection qui doivent alors en tenir compte. Par exemple, si une heuristique vise à maximiser le nombre de modèles, elle risque de privilégier un ensemble non minimal au détriment d'un autre ensemble. Ainsi, **un ensemble de modèles minimal ne contient pas de modèle qui peut être retiré sans changer l'interface multimodale générée**. Si l'environnement contient une seule application, cette propriété peut être approximativement vérifiée ainsi : si l'on retire les modèles de dispositifs de l'ensemble, et que l'on construit

12. Cette définition est sciemment large. En effet, il est difficile de déterminer une propriété calculable qui vérifie de manière exacte si un même action est effectivement exploitée par deux modèles.
13. Un graphe sans cycle est équivalent à un arbre.

un arbre dont la racine est le modèle de l'application, alors toutes feuilles doit avoir au moins une dépendance sur un modèle de dispositif.

4.5.2.1.5 Cohérence

Parmi les propriétés qui viennent d'être définies, seule la validité s'exprime en fonction d'un état d'environnement, les trois autres propriétés sont intrinsèques à un ensemble de modèles. Un ensemble de modèles qui vérifie ces trois propriétés produit une interface utilisateur proche de ce que les concepteurs voulaient exprimer avec les modèles : il est cohérent. Ainsi, **un ensemble de modèles est cohérent s'il est minimal, complet, et si tous ses modèles sont compatibles entre eux**. Des exemples d'ensembles valides, invalides, cohérents et incohérents sont représentés sur la figure 4.12.

La manière de construire tous les ensembles de modèles valides et cohérents est d'abord décrite, puis les heuristiques qui guident les sélections entre ces ensembles sont présentées.

4.5.2.2 Construction d'ensembles valides et cohérents

Les dépôts mettent à disposition du gestionnaire autonomique un ensemble de modèles. À partir de cet ensemble, il est nécessaire de construire des sous-ensembles : les heuristiques utilisées par la suite comparent des ensembles de modèles. Un sous-ensemble doit être cohérents et valides : la composition des modèles du sous-ensemble doit produire une interface utile et utilisable. Par définition d'un ensemble valide, cette construction produit une sélection : les modèles qui n'ont pas d'utilité au regard d'un environnement n'apparaissent dans aucun sous-ensemble.

Les ensembles de modèles valides et cohérents peuvent être générés en calculant l'ensemble des parties de l'ensemble des modèles disponibles, puis en ne gardant que les ensembles valides et cohérents. Cette manière de les générer n'est pas possible en pratique à cause de l'explosion combinatoire. En effet, sa complexité algorithmique est exponentielle en fonction du nombre de modèles [14]. En conséquence, un algorithme naïf plus réaliste est proposé, dont la complexité est moindre en pratique [15].

L'algorithme récursif 4.1 est une manière de construire tous les ensembles valides et cohérents contenant un modèle m s'il est exécuté avec les paramètres $A = \{m\}$, $C = \varnothing$ et $E = \{\varnothing\}$. Le principe est de construire les ensembles en parcourant un graphe connexe de dépendances. Comme le gestionnaire autonomique est écrit avec l'hypothèse que seule une application est présente [16], et que sous cette hypothèse un ensemble valide et cohérent est nécessairement un graphe connexe [17], et que le modèle de l'application fait forcément partie du graphe, l'algorithme construit tous les ensembles valides et cohérents de l'ensemble des modèles disponible si le modèle m passé en paramètre est le modèle de l'application.

L'algorithme 4.1 s'effectue comme suit. À chaque ajout d'un modèle m dans un ensemble C en construction (ligne 3) :

- ajouter tous les modèles dont m dépend dans C (ligne 4) ;
- pour chaque manière de combiner des modèles qui dépendent de m (ligne 5), créer un nouvel ensemble basé sur C (ligne 6) [18].

14. Sa complexité est $O(2^n)$, n étant le nombre de modèles. En effet, si un ensemble a n éléments, l'ensemble des parties de cet ensemble a 2^n éléments. 100 modèles produisent environs 10^{30} ensembles à tester.
15. Mais dont la complexité du pire cas est plus grande.
16. Voir section 4.5 page 70.
17. Voir section 4.5.2.1.2 page 72.
18. En pratique, sont combinés uniquement les modèles qui permettent d'obtenir au moins un ensemble valide et cohérent.

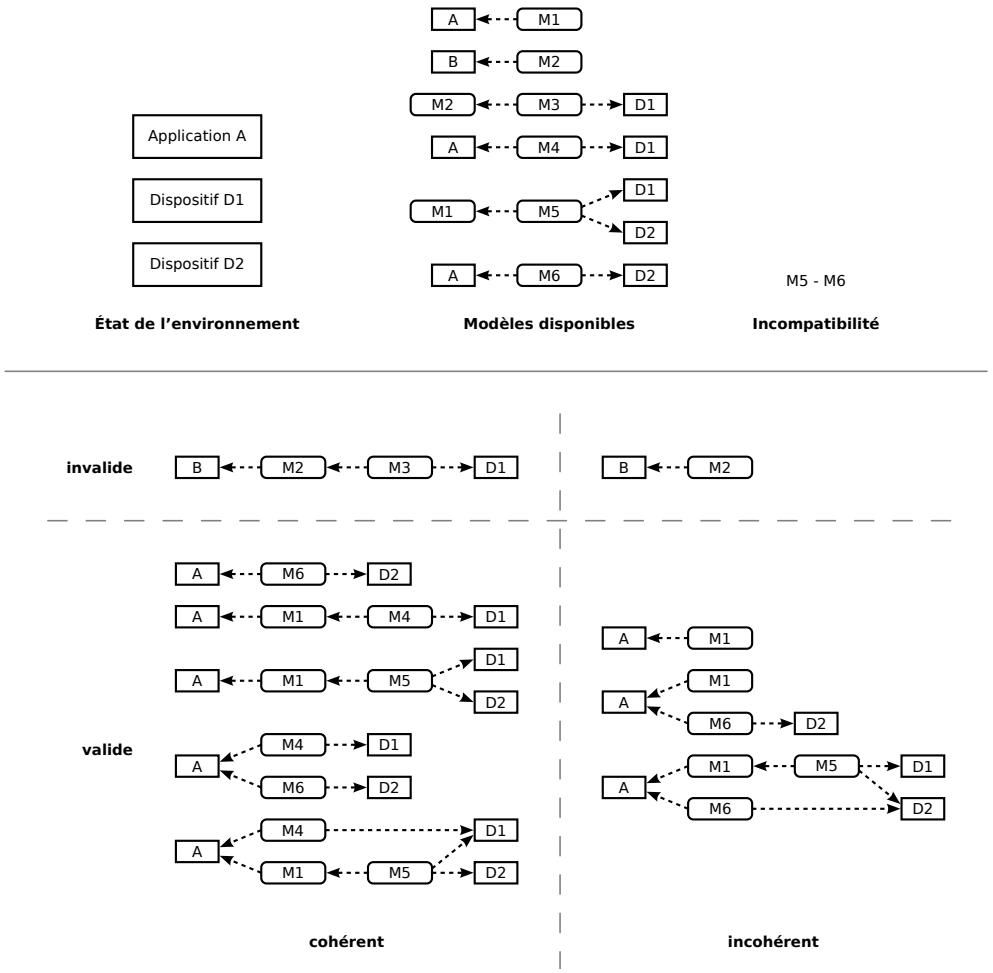


FIGURE 4.12 – Exemples d'ensembles de modèles valides, invalides, cohérents et incohérents. En haut, une application et deux dispositifs sont présents dans l'environnement ; six modèles sont disponibles ; deux modèles sont incompatibles. En bas, quelques ensembles de modèles possibles regroupés par validité et cohérence au sein d'un tableau. {B, M2, M3, D1} et {B, M2} sont invalides car l'application B n'est pas présente dans l'environnement. {B, M2} est de plus incohérent car il ne possède pas de modèle de dispositif : il est donc incomplet. {A, M1} est incohérent pour la même raison. {A, M1, M6, D2} est incohérent car M1 peut être enlevé sans modifier l'interface : il n'est pas minimal. {A, M1, M5, M6, D1, D2} est incohérent car M5 et M6 ne sont pas compatibles.

Cette construction produit tous les ensembles valides et cohérents possibles en fonction de l'état de l'environnement et des modèles disponibles. Nous décrivons maintenant les heuristiques utilisées pour effectuer une sélection parmi ces ensembles.

4.5.2.3 Heuristique : maximisation du nombre de dispositifs

L'objectif de cette heuristique est de préférer les ensembles de modèles qui dépendent de plusieurs dispositifs aux ensembles de modèles qui dépendent de moins de dispositifs : elle maxi-

Algorithme 4.1 Construction d'ensembles valides et cohérents.

Pré-condition : Le graphe des dépendances entre modèles ne contient pas de circuit. $\mathbb{M}$ signifie l'ensemble des modèles possibles.

Paramètres : C est un ensemble de modèles en construction, A un ensemble de modèles à ajouter à C, E est l'ensemble de tous les ensembles de modèles valides et cohérents déjà construits.

Fonctions utilisées : $\mathcal{P}(S)$ renvoie l'ensemble des parties de S, $dep(m)$ renvoie l'ensemble des modèles dont dépend m, $depinv(m)$ renvoie l'ensemble des modèles qui dépendent de m, $valide(S)$ teste la validité de S, $coherent(S)$ teste la cohérence de S.

Post-condition : la valeur de retour contient les ensembles de modèles du paramètre E, et tous les ensembles valides et cohérents contenant les modèles des paramètres C et A.

1: **fonction** CONSTRUIRE($A \subset \mathbb{M}$, $C \subset \mathbb{M}$, $E \subset \mathcal{P}(\mathbb{M})$)
2: **si** $\exists m \in A$ **alors** ▷ test et sélection
3: $C \leftarrow C \cup \{m\}$
4: $A \leftarrow (A \setminus \{m\}) \cup (dep(m) \setminus C)$
5: **pour chaque** $P \in \mathcal{P}(depinv(m) \setminus C)$ **faire**
6: $E \leftarrow E \cup construire(A \cup P, C, E)$
7: **fin pour**
8: **sinon** ▷ $A = \varnothing$
9: **si** $valide(C) \wedge coherent(C)$ **alors**
10: $E \leftarrow E \cup C$
11: **fin si**
12: **fin si**
13: **retourner** E
14: **fin fonction**

mise le nombre de dispositifs. Un exemple d'application de cette heuristique est représenté sur la figure 4.13.

4.5.2.4 Heuristique : maximisation de la multimodalité « conjointe »

Alors que l'objectif de la première heuristique est simplement de privilégier les ensembles dont les modèles couvrent le plus de dispositifs possibles, l'objectif de cette deuxième heuristique est de privilégier les interfaces conçues de manière multimodale par un concepteur. Sans cette heuristique, chaque dispositif peut être utilisé séparément plutôt que conjointement. Lorqu'un concepteur développe un modèle, il a conscience qu'une dépendence ajoutée à son modèle mène vers un dispositif ou une application. Ainsi, lorsqu'il développe une interaction pour plusieurs dispositifs, il déclare au moins deux dépendances qui mènent vers des dispositifs.

Ainsi, cette heuristique compte le nombre de branches supplémentaires créées par des dépendances qui mènent vers des dispositifs. Des exemples d'application de cette heuristique sont représentés sur la figure 4.14.

4.5.2.5 Heuristique : minimisation de la distance sémantique

Le langage de spécification propose l'utilisation d'applications et de dispositifs virtuels [19]. Pour favoriser l'existence de chemins entre les applications et les dispositifs, une bonne pratique

19. Voir la section 4.2.2.2 page 64 et la section 4.3.2 page 67.

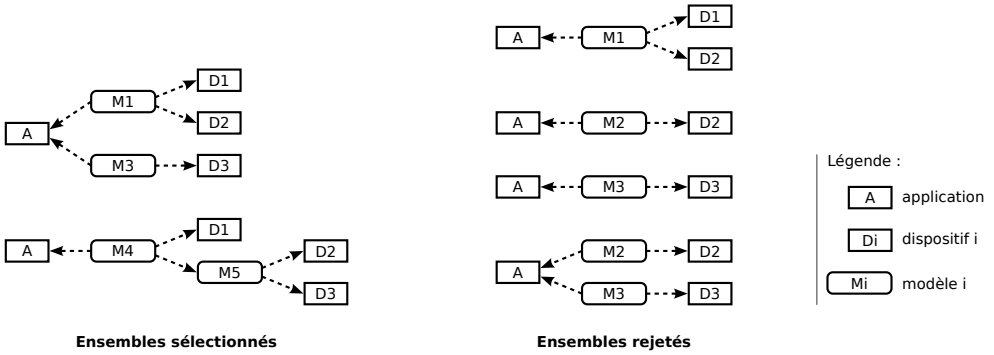


FIGURE 4.13 – Exemple de sélection de modèles avec l'heuristique de la maximisation du nombre de dispositifs. À gauche : deux ensembles sélectionnés qui dépendent de trois dispositifs. À droite : quatre ensembles rejetés car ils dépendent de moins de dispositifs que les deux solutions précédentes. Ainsi, au niveau des modèles, M1, M3, M4 et M5 sont sélectionnés puisqu'ils apparaissent dans les ensembles sélectionnés. Inversement, M2 est rejeté car il n'apparait dans aucun ensemble sélectionné.

consiste à définir des modèles qui décrivent les liens entre les applications virtuelles et les dispositifs virtuels, mais aussi entre les applications virtuelles elles-mêmes et les dispositifs virtuelles eux-mêmes. Si ces modèles entre entités virtuelles ont une utilité indéniable, ils peuvent dans certains cas allonger le nombre de modèles entre un dispositif et une application. Chaque modèle d'entité virtuelle entre un modèle de dispositif et un modèle d'application ajoute une distance sémantique. Le but de cette heuristique est alors de privilégier les ensembles ayant une faible distance sémantique : elle minimise la distance sémantique. Le calcul correspondant effectue la moyenne des tailles de toutes les branches et les ensembles de modèles ayant la plus petite valeur sont privilégiés. Deux exemples d'application de cette heuristique sont représentés sur la figure 4.15.

Notre approche permet soit de définir des fragments d'interface, soit de définir complètement une interface entre des applications et des dispositifs. Une telle interface sur mesure est logiquement plus adaptée qu'un ensemble de fragments mis ensemble. Cette heuristique permet de privilégier ces interfaces complètement définies. En effet, la plus petite distance sémantique correspond à un modèle qui dépend directement de modèles d'application et de modèles de dispositif. Un tel modèle correspond à une interface entièrement spécifiée entre des applications et des dispositifs. Grâce à cette heuristique, un ensemble contenant un tel modèle a de fortes chances d'être sélectionné. Ce cas est représenté sur la figure 4.15, dans l'exemple du bas.

Un allongement de la distance sémantique est représenté dans l'exemple du haut à droite de la figure 4.15. Concrètement, le modèle d'application A peut être le modèle d'un lecteur vidéo particulier, le modèle d'application D1 peut être le modèle de la Wiimote. V1 peut être l'application virtuelle « lecteur multimédia », V2 peut être le dispositif virtuel « manette de jeu ». Il est très probable que l'ensemble en haut à gauche dans lequel la Wiimote est exprimée directement en fonction d'un lecteur multimédia produise une interface plus adaptée que l'ensemble de droite ou les sémantiques des capteurs de la Wiimote sont d'abord exprimées en fonction des capteurs d'une manette de jeu, qui sont ensuite exprimés en fonction d'un lecteur vidéo. L'heuristique amène bien dans ce cas la sélection d'une interface qui est très probablement mieux adaptée.

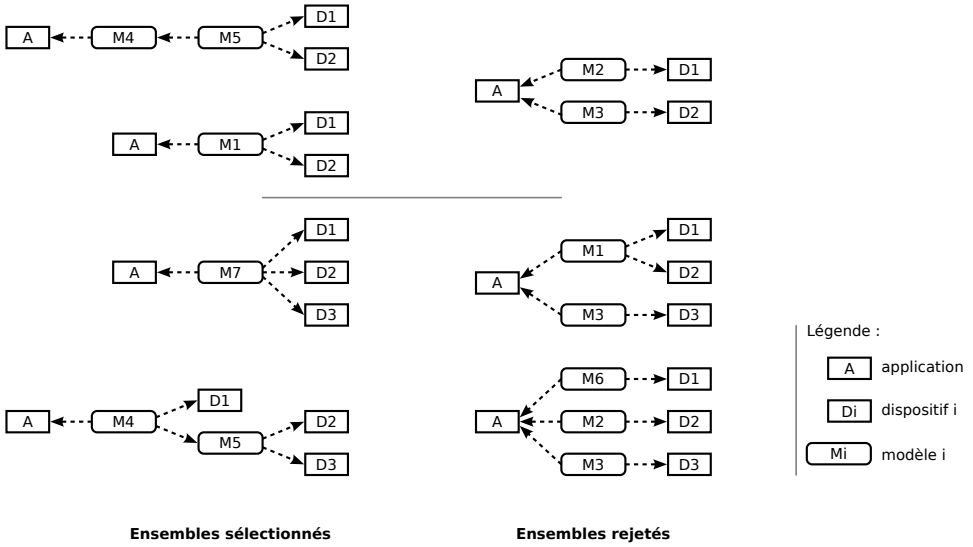


FIGURE 4.14 – Exemples de sélection de modèles avec l'heuristique de la maximisation de la multimodalité « conjointe ». À droite les ensembles sélectionnés, à gauche les ensembles rejetés. En haut : une sélection de deux ensembles de modèles parmi trois. Le nombre pour les ensembles sélectionnés est de 1, le nombre pour l'ensemble rejeté est de 0. En bas : sélection de deux ensembles parmi quatre. Le nombre pour les ensembles sélectionnés est de 2 : 2 en haut, $1 + 1$ soit 2 en bas. Le nombre pour l'ensemble rejeté du haut est de 1 ; celui du bas est de 0.

4.5.2.6 Application des heuristiques

L'utilisation des heuristiques vise à choisir les ensembles de modèles valides et cohérents les plus adaptés à un environnement donné et un utilisateur. Comme le terme « heuristique » l'indique, la solution choisie n'est pas nécessairement la plus optimale. Elles permettent tout au moins d'éviter les interfaces les moins adaptées et de s'exécuter rapidement. D'autres heuristiques peuvent être proposées, notamment si l'hypothèse d'une unique application est retirée. L'utilisateur pourrait alors sélectionner une règle « un dispositif par application », qui viserait à répartir équitablement les dispositifs sur les applications.

Les heuristiques et la manière de les appliquer doivent être différenciées. Le concept d'« heuristique » peut être formulé ainsi : une heuristique est une fonction qui prend un ensemble de modèles en entrée, et qui fournit un nombre en sortie qui reflète approximativement le degré d'adaptation d'une interface qui peut être générée à partir de cet ensemble. Telle que présentée dans les exemples, l'application des heuristiques revient à calculer la fonction de sélection décrite par l'algorithme 4.2 : elle consiste à appliquer successivement une fonction de filtrage paramétrée par une heuristiques h et un ensemble V d'ensembles valides et cohérents. La fonction de filtrage des exemples est notée $filtre : h, V \rightarrow V'$. La fonction consiste à calculer l'ensemble $V' = \{v | \forall v \in V, \nexists w \in V, \ h(w) > h(v)\}$: elle rejette tout ensemble qui ne maximise pas le degré d'adaptation. Ainsi, avec les paramètres $f = filtre$, H l'ensemble ordonné des trois heuristiques présentées précédemment et V tous les ensembles valides et cohérents, l'exécution de l'algorithme renvoie les ensembles de modèles les plus adaptés selon les heuristiques.

Ainsi, les heuristiques et le filtre sont appliqués les uns après les autres : le nombre d'ensembles peut décroitre à chaque étape. D'autres manières d'utiliser les heuristiques sont concevables, mais n'ont pas été implémentées puis testées. Par exemple, toutes les heuristiques pourraient travailler

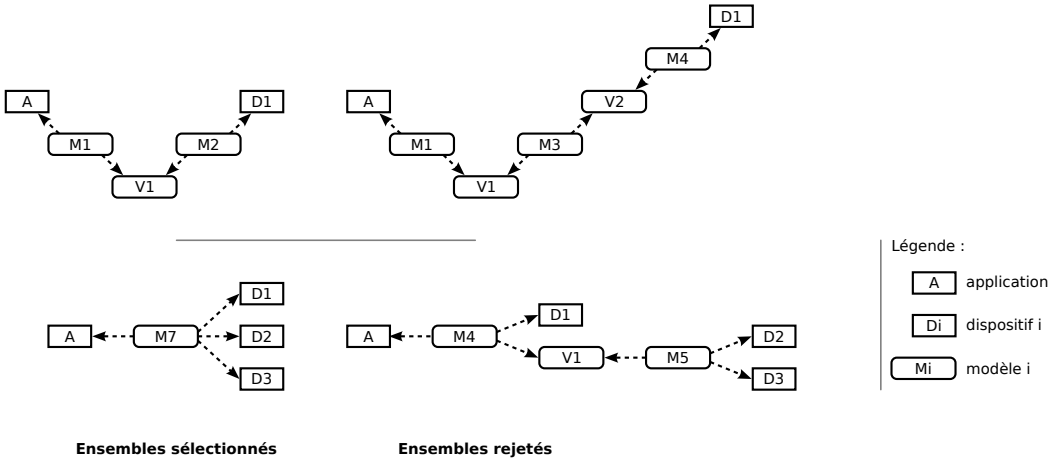


FIGURE 4.15 – Exemples de sélection de modèles avec l'heuristique de la minimisation de distance sémantique. À droite les ensembles sélectionnés, à gauche les ensembles rejetés. En haut : une sélection d'un ensemble parmi deux. La moyenne pour l'ensemble sélectionné est de 5. La moyenne pour l'ensemble rejeté est de 7. En bas : sélection d'un ensemble parmi deux. La moyenne pour l'ensemble sélectionné est de $(3 + 3 + 3)/3$, soit 3. La moyenne pour l'ensemble rejeté est de $(3 + 5 + 5)/3$, soit environs 4,3.

sur les mêmes ensembles, puis la moyenne des degrés d'adaptation, degrés éventuellement pondérés, peut être utilisée pour choisir le meilleur ensemble. La pondération pourrait être réglée par l'utilisateur, via des règles à haut niveau d'abstraction proposées.

Après cette sélection par heuristiques, il reste éventuellement plusieurs ensembles de modèles. Si aucune préférence de l'utilisateur ne contraint à l'utilisation d'un ensemble particulier, un ensemble est choisi aléatoirement. Le choix peut éventuellement être proposé à l'utilisateur s'il a demandé à être prévenu. Le cas échéant, la demande doit être formulée en termes compréhensibles par l'utilisateur, sous forme de questions en langage naturel par exemple. Que le choix de l'ensemble soit aléatoire ou choisi par l'utilisateur, ce choix fait alors partie des préférences de l'utilisateur. Ainsi, si les mêmes ensembles de modèles sont en concurrence, l'utilisateur retrouvera la même interface multimodale.

L'utilisateur peut, pour une certaine application, donner la manière de l'utiliser avec des dispositifs. Pour chaque application, il dispose de la liste des dispositifs virtuels, des applications virtuelles et des dispositifs pour lesquels il existe un modèle d'interaction. Il peut trier cette liste par ordre de préférence. Il a également la possibilité de forcer ce choix, c'est-à-dire qu'avant l'application des heuristiques, seuls les ensembles contenant le modèle préféré seront gardés. S'il n'existe pas d'ensemble le contenant, le second modèle de la liste de préférence est considéré, et ainsi de suite.

4.5.2.7 Synthèse

Trois heuristiques et une manière de les appliquer pour effectuer une sélection ont été présentées pour décrire le fonctionnement du gestionnaire autonomique. Les heuristiques présentées devraient être testées et améliorées avec de nombreux cas réels. Les algorithmes liés à ces heuristiques sont très simples à implémenter : il s'agit essentiellement de parcours d'arbre. L'application de chaque heuristique présentée a une complexité algorithmique en temps majorée par $O(n * m)$,

Algorithme 4.2 Sélection d'ensembles par filtrage basé sur des heuristiques.

Pré-condition : H un ensemble ordonné d'heuristiques, f une fonction de filtrage qui rejette les ensembles les moins adaptés selon une heuristique, V un ensemble d'ensembles de modèles cohérents.

Post-condition : la valeur de retour contient un sous-ensemble de V contenant les ensembles de modèles les plus adaptés

```
fonction SELECTION(f, H, V)
    si ∃h ∈ H alors                                  ▷ test et sélection de la 1re heuristique
        retourner selection(f, H \ {h}, f(h, V))
    sinon                                            ▷ H = ∅
        retourner V
    fin si
fin fonction
```

n étant le nombre d'arbres et m la complexité du parcours de l'arbre [20], cette complexité parait faible au regard de l'importance du travail effectué. La rapidité, la simplicité, et le pouvoir d'expression relativement puissant rendent les heuristiques intéressantes pour l'aide à la sélection automatique de modèles. Il existe plusieurs manières de les utiliser pour effectuer une sélection, trouver la meilleure manière reste une question ouverte.

Les définitions de la validité et de la cohérence sur les ensembles de modèles permettent la sélection des ensembles qui peuvent être transformés en interface utilisateur utilisable. Cette préparation permet aux heuristiques d'être exprimées simplement en sélectionnant des ensembles sous forme d'arbre. Cette préparation permet également aux heuristiques de travailler rapidement : la réduction du nombre d'ensembles possibles est drastique. La manière de calculer les modèles incompatibles pourrait cependant être affinée pour retenir des ensembles pouvant aboutir à une interface utilisable mais qui sont pour l'instant rejetés.

La sélection se divise en deux étapes : construction d'ensembles valides et cohérents puis sélection d'ensembles via applications successives d'heuristiques. Cette section a souligné la pertinence de cette séparation.

La sélection produit un unique ensemble de modèles prêts à être composés et transformés. La composition et la transformation de modèles est l'objet de la section suivante.

4.5.3 Composition et transformation de modèles

La composition et la transformation de modèles visent à produire un modèle de chaine de médiation qui peut être instancié. Les modèles à composer et à transformer sont issus d'un ensemble valide et cohérent obtenu par l'étape de sélection. Alors que l'étape de sélection travaille essentiellement au niveau des modèles et de leurs dépendances, cette étape de composition et transformation travaille au niveau des composants et des liaisons.

La composition, puis les transformations sont présentées.

4.5.3.1 Composition

La composition vise à obtenir un unique modèle à partir de l'ensemble de modèles construit par l'étape de sélection : les dépendances disparaissent et seules restent des instances de compo-

20. Un parcours en profondeur a une complexité linéaire en fonction du nombre de sommets.

sants et des liaisons entre les ports de ces instances.

Au sein d'un modèle, une liaison peut être exprimée en fonction d'un port d'une instance de composant déclarée dans le modèle, et d'un port d'une instance déclarée dans l'une de ses dépendances. La manière de construire cet ensemble de modèles assure que si un modèle est dans cet ensemble, alors tous les modèles dont il dépend sont également dans cet ensemble [21]. Cette propriété garantit que toutes les liaisons référencent des ports existants.

La composition est très simple : elle consiste à assembler les liaisons et les instances de composant de tous les modèles. Une fois la composition effectuée, le modèle produit peut être transformé.

4.5.3.2 Transformation

Cette transformation est utile pour deux raisons. Premièrement, les modèles créés par un concepteur d'interaction ne requièrent pas certains détails dans le but de simplifier leur création : absence de typage de données circulant entre les ports, et absence de contrainte sur les d'intervalles de valeur requis ou fournis par les ports. Cette première raison oblige à ajouter des informations pour que l'exécution fonctionne correctement. Deuxièmement, les composants représentant des dispositifs peuvent avoir des ports non utilisés, il est alors possible d'exploiter ces ports libres pour ajouter des possibilités à l'interface utilisateur.

Les quatre sections qui suivent expliquent ces deux transformations.

4.5.3.3 Transformation : exploitation des ports libres des dispositifs et des applications

Suite à la composition des modèles, il peut rester des ports de mandataires de dispositif qui ne sont liés à aucun autre port. Ces ports libres peuvent alors être utilisés. Une recherche des possibilités de connexion est menée, et les liaisons possibles sont ajoutées. Un ensemble de règles est appliqué pour réaliser ces connexions. Dans l'ordre de leur application :

- Les ports libres du mandataire d'application sont considérés en priorité. Pour chaque port libre, une recherche est effectuée sur l'ensemble des ports libres des mandataires de dispositifs. Si les ports sont compatibles, une liaison est créée.
- Pour chaque port non libre, une recherche est effectuée sur l'ensemble des ports libres des mandataires des dispositifs. S'ils sont compatibles, alors une liaison est créée.

Deux ports sont compatibles si l'un est un port d'entrée et l'autre un port de sortie. De plus, les types de données qui transitent par eux doivent être compatibles, c'est-à-dire que soit les types de données sont identiques, soit il existe un composant capable de convertir les types du port de sortie vers les types du port d'entrée [22]. L'application de ces règles se fait de manière à répartir le plus équitablement possible les capteurs sur les tâches afin d'éviter de lier tous les capteurs sur une seule tâche.

Si un port d'entrée requiert une liste, la recherche privilégie un port de sortie qui émet une liste compatible. En l'absence d'un tel port, une recherche de ports compatibles avec chacun des éléments de la liste est effectuée. Si pour chaque élément de la liste il existe un port compatible, alors un composant de fusion est ajouté pour rendre compatible cet ensemble de ports avec le port d'entrée.

Par l'application de ces deux règles, des interfaces utilisables peuvent être générées, néanmoins des interfaces peuvent également être en partie inutilisables. Deux raisons à cela : premièrement,

21. Voir section 4.5.2.2 page 74.

22. La compatibilité entre deux types est un peu plus complexe qu'une relation binaire, de manière à préférer par exemple la liaison entre deux ports traitant des entiers plutôt qu'une liaison entre un port fournissant un flottant et un port acceptant des entiers.

un capteur peut envoyer des données en continu alors qu'une tâche élémentaire requiert une réception de données non continue [23] ; deuxièmement, plusieurs capteurs peuvent envoyer des données corrélées [24]. Pour contourner le premier problème, si un port de sortie envoie des données en continu et que le port d'entrée nécessite seulement des valeurs non continues une valeur de ce flux peut être explicitement sélectionnée par l'utilisateur, en appuyant sur un bouton par exemple. Pour contourner le second problème, si un ensemble de capteurs produit des valeurs corrélées, alors dans l'ensemble des ports qui les représentent au plus un port peut ne pas être libre.

Pour obtenir des interfaces plus utilisables, il est utile de tenir compte de la proximité sémantique des capteurs au sein d'un même dispositif. La proximité sémantique est définie informellement par l'association cognitive qu'un utilisateur peut faire entre des capteurs ou des ensembles de capteurs : par exemple, un bouton « + » et « - » sont proches sémantiquement, de même que deux boutons « haut » et « bas » d'une part, et « droite » et « gauche » d'autre part, ainsi que les ensembles {« haut », « bas »} et {« gauche », « droite »}. Cette proximité sémantique est souvent accompagnée d'une proximité spatiale des capteurs qui renforce l'association cognitive qu'un utilisateur peut faire. Similairement, des tâches élémentaires d'une application peuvent être groupées. Un exemple de groupement sémantique de capteurs est montré sur la figure 4.16. La création de liaisons entre les capteurs et les tâches d'une application peut être guidée par ces groupements : sont privilégiées les liaisons entre les éléments des groupements similaires. La totalité des groupements au sein d'un environnement peut être assimilé à un arbre dont les noeuds représentent les groupements et les feuilles les capteurs. La racine représente le groupement de tous les groupements existants dans un environnement donné, chaque nœud de premier niveau correspond à un dispositif présent dans l'environnement, et ainsi de suite jusqu'aux capteurs. C'est logiquement au niveau d'un groupement de capteurs que la propriété de corrélation peut être appliquée.

Prendre en compte le type de capteur, c'est-à-dire la syntaxe et la sémantique de la donnée qu'il fournit, permet également d'obtenir des interfaces plus utilisables. Par exemple, un module de reconnaissance de geste peut être prévu pour fonctionner avec des valeurs issues d'un accéléromètre tridimensionnel. Lier ce module à trois sources de données qui ne viennent pas d'un accéléromètre produit localement une interface inutile.

4.5.3.4 Transformation : adaptation des types de données et des intervalles de valeurs

Une tâche élémentaire d'une application requiert un type de données particulier. Si elle reçoit une donnée dont le type n'est pas identique, une erreur se produit. Cependant, des types de données peuvent être compatibles, c'est-à-dire qu'il existe un mécanisme de conversion systématique qui permet à une valeur dans un type donné d'obtenir une valeur utile dans un type de donnée compatible, moyennant éventuellement une perte de précision. C'est typiquement le cas des entiers et des flottants. Puisque le concepteur d'interaction ne se soucie pas des types de données lorsqu'il modélise une interface, et que l'étape d'exploitation des ports libres peut relier des ports compatibles mais dont les types de données diffèrent, il est nécessaire d'ajouter des conversions de données au sein d'une chaine de médiation. Un ensemble de composants de médiation existe, et des composants peuvent être insérés par le gestionnaire autonomique entre

23. Par exemple, un gyroscope envoie des données en continu, relié à une tâche élémentaire de contrôle du niveau sonore, maintenir un certain niveau sonore requiert de maintenir le capteur dans une certaine position.

24. Par exemple, un accéléromètre tridimensionnel peut être vu comme un ensemble de trois capteurs qui fournissent trois flux des données corrélées : il est difficile pour un utilisateur de n'agir que sur un seul flux.

FIGURE 4.16 – Exemple de 8 groupements sémantiques. En haut, une manette de jeu, en bas des groupements sémantiques possibles des capteurs représentés par les courbes fermées en pointillés.

des ports pour effectuer ces conversions. Les compatibilités entre les types sont résumées par le tableau 4.1.

En plus du type de donnée, une tâche élémentaire peut n'accepter des valeurs que sur un intervalle donné. Par exemple, une tâche élémentaire de contrôle de niveau sonore peut accepter seulement des valeurs comprises dans l'intervalle [0, 100]. Un gyroscope peut fournir des valeurs comprises dans l'intervalle [-180, +180]. Si les valeurs issues du gyroscope sont directement envoyées à la tâche, une anomalie [25] peut se produire. Pour résoudre ce problème, comme pour les types de données, le gestionnaire autonomique insère un composant de médiation convertissant les données entre les ports. Ce convertisseur implémente une transformation affine [26] de l'ensemble des valeurs de l'intervalle du port de sortie vers l'ensemble des valeurs de l'intervalle du port d'entrée.

Les types de données et les intervalles sont propagés à travers les instances de composants, de manière à détecter les incompatibilités entre deux ports même s'il existe des composants intermédiaires. Pour permettre ce mécanisme, les composants déclarent les propagations de types et d'intervalles entre ports d'entrée et ports de sortie.

4.5.3.5 Transformation : adaptation à un dispositif universel

Un dispositif universel est un dispositif capable de fournir tous les types de données possibles. Sa différence est tellement marquée par rapport aux autres dispositifs qu'il est considéré à part. Un dispositif capable d'afficher une interface graphique adaptable à l'exécution est généralement un dispositif universel : cette interface peut proposer un bouton pour permettre à l'utilisateur de produire une donnée de type *évènement*, un curseur [27] pour produire un entier ou un flottant,

25. Selon la conception de l'application, ce cas de figure peut être géré, conduire à un état indéterminé, provoquer une erreur ou un arrêt brutal, etc.

26. Une fonction affine peut s'écrire sous la forme $y = ax + b$, avec a et b constants. Il existe une méthode pour obtenir a et b à partir de deux couples de valeurs (x, y) connus. Dans notre cas, le couple des minimums des intervalles ainsi que le couple des maximums sont considérés.

27. *Slider* en anglais.

	évènement	entier	flottant	chaine	sélection	liste
évènement	✓					
entier	✓	✓	✓		✓	
flottant	✓	✓	✓		✓	
chaine	✓			✓		
sélection	✓	✓	✓		✓	
liste	✓					✓

Table 4.1 – Compatibilité entre les types de données. Une ligne représente le type d'un port de sortie, une colonne représente le type d'un port d'entrée, une case cochée représente une compatibilité entre le type d'un port de sortie et le type d'un port d'entrée. Par exemple, la première ligne se lit ainsi : un port de sortie dont le type est *évènement* est compatible uniquement avec un port d'entrée dont le type est *évènement* ; la première colonne se lit ainsi : un port d'entrée dont le type est *évènement* est compatible avec tous les ports de sorties.
Le type *chaine* correspond aux chaines de caractères. Le type *sélection* est une valeur de chaine de caractère parmi un ensemble ordonné de chaines de caractères. Le type *liste* est un ensemble ordonné de types. Deux listes sont compatibles uniquement si elles ont le même nombre d'éléments et si chaque élément à une certaine position est compatible avec l'élément de l'autre liste situé à la même position.

un menu déroulant pour produire une *sélection*, etc. Un dispositif de reconnaissance de la parole est également un dispositif universel.

Le composant mandataire d'un tel dispositif a la particularité d'avoir un seul port, de permettre la création de port, et d'appliquer une règle différente pour le calcul des modèles incompatibles. Si le port est libre, la transformation consiste à créer une liaison entre ce port et chaque port d'entrée de l'application, et à lui fournir une configuration particulière.

Chaque port de composant possède une chaine de caractères décrivant son rôle de manière compréhensible par un utilisateur. Dans le modèle d'une application, à chaque port d'entrée est associée une chaine décrivant la tâche de manière compréhensible par un utilisateur. Cette chaine descriptive est utilisée librement par le dispositif : une interface graphique générée peut l'utiliser comme un intitulé pour un composant graphique, un dispositif de reconnaissance de la parole peut l'utiliser comme lexique à reconnaitre [28]. Nous considérons un exemple : une application possède une tâche de réglage de volume qui accepte des entiers dans [0,100], la chaine la décrivant s'appelle « volume ». Sur l'écran d'un smartphone est alors créé un curseur dont l'intitulé est « volume » et qui autorise le mouvement du curseur entre les valeurs 0 et 100 ; à chaque fois que le curseur est déplacé, le dispositif envoie la nouvelle valeur à la tâche de réglage de volume. De même, un dispositif de reconnaissance de la parole se configure pour reconnaitre le mot « volume » puis un nombre entre 0 et 100. À chaque fois que ce mot et un nombre sont reconnus, le dispositif envoie le nombre est envoyé à la tâche de réglage de volume.

Enfin, toujours sous la condition que le port était libre, pour permettre le routage correcte des données, il est nécessaire de redéfinir les liaisons avec le port. Un nouveau port est ajouté au mandataire pour chaque liaison. Une configuration est ajoutée pour que le mandataire connaisse la correspondance entre un type de données à fournir, une chaine descriptive et un de ses nouveaux ports de sortie sur lequel envoyer une donnée.

28. Deux modèles sont donc incompatibles s'ils possèdent chacun une liaison avec le port et si leur description est identique.

La notion de dispositif universel lorsque le dispositif repose sur une interface graphique s'apparente aux travaux en IHM sur la génération d'interfaces graphiques à partir d'un arbre de tâches. Ces travaux sont anciens, l'exemple séminal étant le système ADEPT [Johnson *et al.*, 1993]. Plus récemment, ces travaux ont été repris et étendus dans le cadre de l'ingénierie dirigée par les modèles pour la plasticité des interfaces [Thevenin et Coutaz, 1999], qui constitue aujourd'hui un domaine actif en IHM.

Dans ce contexte très réactif, nos travaux ne visent pas à générer l'interface graphique d'une application mais uniquement et plus simplement un dispositif logique pour spécifier des entrées à une application qui contient par ailleurs son interface de sortie. Nous n'avons pas à traiter l'enchainement des tâches (onglets, fenêtre...) ni le regroupement logique de tâches à partir de la hiérarchie des tâches.

4.5.3.6 Transformation en une chaine de médiation

La transformation finale consiste à convertir le modèle d'interaction basé sur des composants en un modèle de chaine de médiation basé sur des médiateurs. Chaque configuration par défaut est éventuellement réécrite pour tenir compte des configurations de l'utilisateur. Les instances de composants sont converties en instances de médiateurs, les liaisons entre ports d'instances de composants sont converties en liaisons entre ports d'instances de médiateurs. Les modèles de mandataire deviennent des modèles d'instance de médiateur « frontière » capables d'assurer la communication entre une chaine de médiation et des instances de services présentes dans la plateforme à services. Trois exemples de ces instances de médiateur sont représentés sur la figure 4.8 page 70 : ils se situent dans la plateforme de médiation et communiquent avec les mandataires ; ils sont situés à l'extrême droite et l'extrême gauche de la chaine de médiation.

Une fois cette dernière transformation effectuée, le modèle de chaine de médiation est alors géré par la plateforme de médiation.

4.6 Synthèse

Ce chapitre a présenté notre proposition conceptuelle pour la construction d'interfaces multimodales dans les environnements pervasifs. Cette proposition couvre la spécification et l'exécution d'une interface multimodale. La spécification est réalisée par un concepteur d'interfaces qui modélise une interface multimodale. Ces spécifications peuvent être complètes ou partielles. L'exécution est réalisée par une plateforme multimodale constituée de deux parties distinctes : un gestionnaire autonomique et une plateforme d'intégration. Le gestionnaire autonomique est en charge d'utiliser toutes les informations à sa disposition pour créer un modèle d'interface multimodale adaptée à l'environnement. La plateforme d'intégration est en charge d'exécuter ce modèle et d'assurer la communication entre les applications, les interfaces multimodales et les dispositifs.

Le langage proposé permet la spécification d'interfaces multimodales par un concepteur d'interaction grâce à des concepts d'interface utilisateur. Ce langage résulte d'un compromis entre la facilité pour un concepteur de spécifier une interface et la réalisation du gestionnaire autonomique qui est chargé de transformer les spécifications en une interface exécutable. Le langage est basé sur l'approche à composants comme beaucoup de langages dédiés aux interfaces multimodales. Comme souligné pour la plateforme OpenInterface [29], cette approche a l'avantage de ne

29. Voir la section 3.2.4 page 55.

pas trop s'éloigner du modèle de l'exécution, de permettre des spécifications relativement fines, de pouvoir représenter la spécification graphiquement de manière directe avec des boites et des traits, et d'être assimilable relativement simplement par des personnes non spécialistes de la programmation. En effet, les concepts sont limités : une spécification d'interface est un assemblage de composants qui repose sur seulement trois concepts :

- les instances de composants configurables qui réalisent une fonction ;
- les liaisons entre ports qui définissent les chemins que doivent suivre des données ;
- les dépendances vers d'autres assemblages.

Les concepts sous-jacents de type de données, d'intervalles de valeur et de transmission continue ou non sont masqués, mais peuvent éventuellement être considérés par un concepteur pour contrôler plus finement les interfaces.

Les dispositifs et applications virtuels sont des conventions proposées par le langage aux concepteurs, qui leur permettent de décrire le comportement des dispositifs et des applications en fonction de classes d'applications ou de classes de dispositifs. Ces conventions s'intègrent parfaitement dans le langage : en effet, ils constituent des assemblages dont les ports ont une signification partagée par les concepteurs. Ils sont néanmoins centraux dans notre approche : ils permettent de spécifier des fragments d'interfaces sans connaitre l'environnement dans lequel ils seront utilisés.

Le concept de classes d'applications et de dispositifs n'est pertinent que si une classe contient un nombre réduit de concepts : elle garde alors une généralité qui lui permet d'être la cible d'une projection de nombreux dispositifs et applications. Cette proposition vise alors en premier lieu l'intégration de tous les dispositifs dans les environnements pervasifs, car ils produisent toujours un faible nombre de données de natures différentes.

Le principal avantage par rapport aux plateformes existantes dédiées aux interfaces multimodales [30] est la capacité à adapter automatiquement une interface multimodale à un environnement. Dans les approches existantes, le cas général est un environnement connu à la conception, homogène et statique [31]. Dans notre approche, le cas général est un environnement inconnu à la conception, hétérogène et dynamique ; un environnement connu, homogène et statique constitue seulement un cas particulier. La plateforme à services gère l'hétérogénéité en définissant une vue homogène de l'environnement. La plateforme de médiation adapte les données qui circulent entre les dispositifs et les applications. Le gestionnaire autonomique contrôle la plateforme de médiation pour l'adaptation à l'environnement. La possibilité de construire des interfaces multimodales au sein d'un environnement hétérogène, dynamique et non prévu à la conception nous permet d'avancer que notre proposition répond à certaines caractéristiques primordiales des environnements pervasifs [32].

L'apport de notre solution par rapport aux plateformes généralistes dédiées aux environnements pervasifs [33] est sa prise en compte des caractéristiques des interfaces multimodales. Dans notre proposition, la plateforme d'intégration est généraliste : c'est la manière dont sont conçus le gestionnaire autonomique et les modèles – des descriptions d'interaction, de dispositifs et d'applications – sur lesquels il travaille qui spécialise notre approche aux interactions multimodales. Remplacer tout ou partie du gestionnaire autonomique et les modèles pourrait à priori spécialiser notre approche dans un autre domaine. La spécialisation du gestionnaire autonomique aux interfaces multimodales repose sur :

30. Voir la section 3.2 page 49.
31. Certaines proposent des mécanismes pour le dynamisme ou l'hétérogénéité.
32. Comme expliqué au chapitre 3 page 37.
33. Voir la section 3.1.5 page 44.

– la distinction entre dispositifs et applications ;
– le concept de dispositif universel ;
– la présence et l'utilisation de chaines décrivant chaque port ;
– les heuristiques guidant la sélection de modèles ;
– les règles de transformation de modèles ;
– la manière pour un utilisateur de configurer une interface.

Le gestionnaire autonomique permet ainsi la transformation de concepts d'interfaces multimodales fournis par le langage d'interaction vers les concepts propres à l'exécution d'une interface multimodale fournis par la plateforme d'intégration. Ce chapitre a présenté de manière détaillée cette transformation. La transformation inverse – des concepts de l'exécution vers les concepts d'interaction – n'a pas été suffisamment étudiée pour constituer une partie solide de notre proposition. L'étude de cette transformation devrait être particulièrement utile pour l'utilisateur final : la représentation simple et compréhensible d'une interface utilisateur faciliterait son appréhension de l'interface et l'expression de ses préférences. Du couplage de ces deux transformations pourrait émerger le concept simple et puissant de modèle de l'exécution : un modèle qui est le miroir du comportement et de l'état de l'exécution et vice-versa ; toutes modifications dans l'état et le comportement de l'exécution peuvent être répercutées dans le modèle, toutes modifications du modèle peuvent être répercutées dans l'état et le comportement de l'exécution. Ce concept est connu sous le terme de *model at runtime* [Blair *et al.*, 2009].

Tandis que ce chapitre a présenté notre solution conceptuelle d'une plateforme pour la multimodalité dans les environnements pervasifs, nous présentons dans le chapitre suivant une implémentation logicielle de cette solution.

Chapitre 5

DynaMo : une plateforme pour la multimodalité en environnement pervasif

Ce chapitre donne un aperçu de la mise en œuvre logicielle de la proposition présentée dans le chapitre précédent. Cette réalisation technique et la réflexion conceptuelle ont été menées conjointement. Le résultat a été nommé DynaMo, pour *dynamic modalities*.

La conclusion du chapitre 3 énonce deux pistes possibles pour réutiliser les outils existants : partir d'un outil dédié aux interfaces multimodales, ou partir d'un outil fournissant certaines propriétés pour la gestion de l'hétérogénéité et le dynamisme. La dernière piste a été choisie : un travail de recherche dans le cadre d'un doctorat nécessite une certaine liberté qui peut être limitée par une vision déjà promu par des outils trop proches du champ de recherche. Modifier ou adapter un outil existant – éventuellement en profondeur – pose un risque non négligeable de se retrouver avec de sérieuses complications techniques compromettant l'avancée conceptuelle. Construire au-dessus d'un outil existant limite fortement ce risque, mais il peut en résulter un « isolement » conceptuel.

Ainsi, deux importantes briques logicielles ont été utilisées : iPOJO [1] et Cilia [2]. iPOJO est le modèle à composant dans lequel sont programmés les mandataires. Cilia constitue la majeure partie de la plateforme de médiation. iPOJO et Cilia reposent sur OSGi [3] qui fournit logiquement la majeure partie de la plateforme à services. Contrairement à d'autres technologies à services, OSGi met l'accent essentiellement sur l'utilisation au sein d'une même machine. Ce chapitre présente une manière théorique de gérer un environnement distribué.

Ce chapitre se découpe en trois parties qui reprennent partiellement le découpage de la proposition : le langage de spécification pour les interfaces multimodales, la plateforme d'intégration et enfin le gestionnaire autonomique.

1. Voir la section iPOJO page 41.
2. Voir la section Cilia page 42.
3. Voir la section OSGi page 40.

5.1 Langage de spécification pour les interfaces multimodales

Le langage de spécification pour les interfaces multimodales est décrit sous la forme d'un métamodèle de classe. Ce métamodèle proposé au concepteur est qualifié de langage car il contient les concepts et en partie la manière de les manipuler : créer un modèle conforme au métamodèle est la manière pour le concepteur de mettre en œuvre ses connaissances en interaction.

Aborder uniquement le langage – le métamodèle – utilisé par le concepteur ne permet pas d'illustrer pleinement la proposition. C'est pourquoi deux autres métamodèles sont également présentés. Ces métamodèles se recouvrent partiellement, mais la terminologie et la structure changent : le concepteur d'interaction utilise un métamodèle proche des interactions multimodales, le programmeur utilise un métamodèle proche du code qu'il écrit, le gestionnaire autonomique utilise un métamodèle proche de la médiation de données.

Ces trois acteurs ont chacun un point de vue sur les modèles de DynaMo :

- Le point de vue du concepteur d'interaction est orienté vers la spécification d'interactions multimodales : son objectif est de créer des interactions entre dispositifs et applications sous forme d'assemblages d'instances de composants configurables.
- Le point de vue du développeur : il programme les médiateurs Cilia, qu'il doit par la suite modéliser sous forme de composants configurables. De même, il programme les mandataires sous forme de composants iPOJO, qu'il doit par la suite modéliser sous forme de composant. Ces deux modélisations visent à donner à DynaMo toutes les informations nécessaires pour qu'il puisse exploiter les médiateurs Cilia et les composants iPOJO disponibles.
- Le point de vue du gestionnaire autonomique : son but est d'assembler et transformer des modèles d'interaction pour obtenir un modèle de chaine de médiation. Un modèle d'interaction est constitué d'un ensemble d'instances de composant, d'un ensemble de liaisons et d'un ensemble de dépendances.

Chaque métamodèle est adapté à l'acteur auquel il s'adresse. Les conversions systématiques entre les modèles sont possibles.

La suite de cette section présente ces métamodèles. Ils sont tous représentés en détail dans l'annexe A page 128.

5.1.1 À propos de la notation graphique

Les métamodèles sont représentés sous forme de diagrammes de classes UML [uml241, 2011]. Les boites représentent des classes, les traits représentent des relations entre classes. Le triangle vide représente la généralisation, les losanges pleins représentent la composition. La multiplicité dénotée par l'étoile seule correspond à [0..*]. L'étoile employée comme borne supérieur d'une multiplicité dénote l'absence de borne.

Tous les métamodèles sont sous la forme d'une arborescence de composition. Cela permet de les stocker facilement en XML.

Pour garder la présentation des métamodèles concise, les attributs de classe qui ne soient pas nécessaires pour la compréhension des métamodèles ne sont pas présentés. C'est le cas de la plupart des identifiants et de quelques autres attributs.

5.1.2 Métamodèle de conception

Ce métamodèle est présenté progressivement. Il est représenté complètement dans l'annexe A page 129. Il définit les modèles que peut produire le concepteur, mais aussi une partie des

informations auxquelles il a accès, et qui doivent être présentées par un outil de conception.

Le concepteur d'interaction manipule trois entités : les applications, les dispositifs et les composants configurables. Ces trois entités sont représentées sur la figure 5.1. Les applications et les dispositifs sont éventuellement virtuels : ils sont alors une convention entre tous les concepteurs. Chaque entité peut envoyer ou recevoir des données. Une **application** dispose de tâches qui reçoivent des données pour effectuer un traitement, elle rend certaines informations accessibles qui peuvent être envoyées. Un **dispositif** dispose de capteurs qui produisent des données et d'effecteurs qui reçoivent des données et réagissent en conséquence. Un **composant configurable** implémente une fonction qui effectue des calculs sur des données qui arrivent sur des ports d'entrée et émet les résultats sur des ports de sorties.

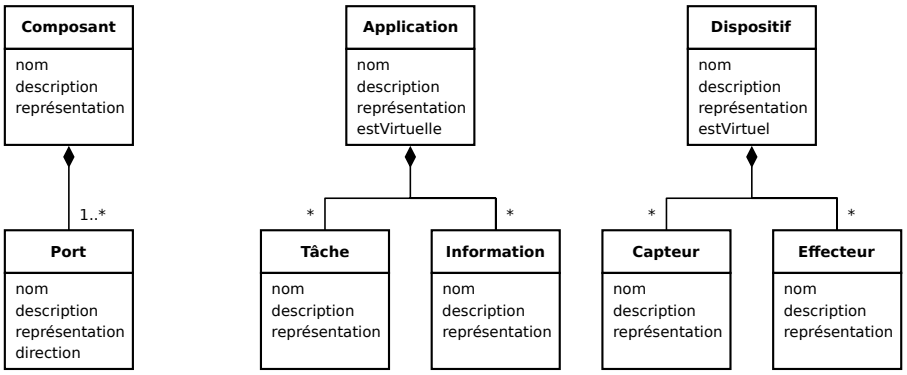


FIGURE 5.1 – Métamodèle des applications, dispositifs et composants.

Toutes les classes disposent d'un nom et d'une description qui permettent au concepteur de connaitre et de comprendre leur utilité. Elles contiennent éventuellement une représentation graphique qui permet à un outil de conception de les afficher sous forme d'icône par exemple.

Une application représente au sens large tout ce qui peut être commandé par un utilisateur final : c'est une extrémité d'une interface multimodale. Un dispositif représente un moyen de rendre compte de l'état du monde physique sous forme de données, et d'agir sur l'état du monde physique pour transmettre de l'information à l'utilisateur final : c'est une autre extrémité d'une interface multimodale. Un composant représente une fonction qui peut être utile dans le processus d'adaptation des données entre les applications et les dispositifs.

Une instance de composant, d'application ou de dispositif est créée à partir d'un modèle préexistant : en déclarant quelles entités il souhaite manipuler, le concepteur crée des instances qui sont automatiquement remplies. Ces instances ne sont pas modifiables dynamiquement : il ne peut pas ajouter de ports à un composant, attribuer un capteur à un autre dispositif, changer les noms ou les descriptions, etc.

Il peut par contre configurer un composant. La configuration d'un composant est représentée sur la figure 5.2. Une configuration est un ensemble de propriétés prédéfinies pour chaque composant. Chaque propriété a un nom, une description et une valeur par défaut. Le concepteur peut modifier la valeur selon ses besoins. Il peut également réinitialiser la valeur à la valeur par défaut. La valeur doit être d'un certain type – les types sont détaillés plus loin. Un outil pour la spécification d'interfaces peut utiliser le type pour vérifier la validité de la valeur saisie, ou pour produire un composant graphique adapté à la saisie de telles valeurs.

Pour unifier le traitement les composants, applications et dispositifs d'une part, et les ports, tâches, informations, capteurs et effecteurs d'autre part, deux généralisations sont définies : les

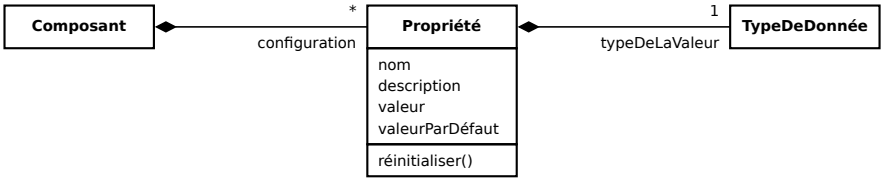
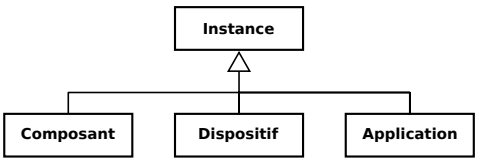
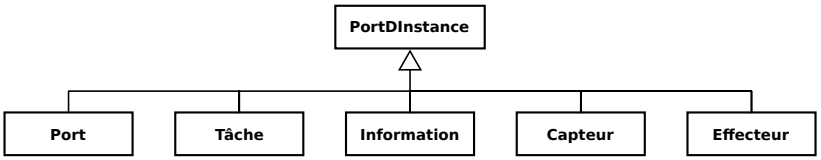


FIGURE 5.2 – Métamodèle des configurations.

instances et les ports d'instances. Ces deux généralisations sont représentées sur la figure 5.3. Bien que les notions d'instances et de ports d'instances ne sont pas présentées au concepteur, les représentations graphiques de toutes les instances peuvent être similaires, idem pour les ports d'instances.



(a) Généralisation : les instances.



(b) Généralisation : les ports d'instances

FIGURE 5.3 – Généralisation des composants, applications et dispositifs, et généralisation des ports, tâches, informations, capteurs et effecteurs.

Cette modélisation peut paraitre contre-intuitive – un port d'instance est la généralisation d'un port – uniquement parce que les noms des classes ont été choisis avec les concepts du domaine des interactions.

Ces généralisations permettent de définir très simplement une interaction : une interaction est un ensemble d'instances et un ensemble de liaisons entre les ports des instances. Le métamodèle des interactions est représenté sur la figure 5.4. Une liaison a évidemment la contrainte de ne pouvoir lier deux ports qui émettent des données ou deux ports qui reçoivent des données.

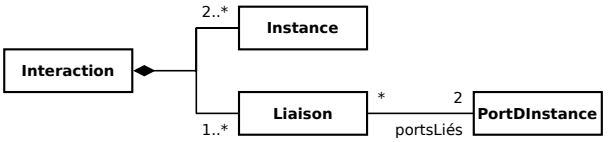


FIGURE 5.4 – Métamodèle des interactions.

Une interaction s'exprime toujours en fonction de dispositifs et d'applications. Ainsi, une interaction contient toujours une application et au moins un dispositif – virtuels ou non, ou d'une application et une application virtuelle, ou d'au moins un dispositif et un dispositif virtuel. Cette contrainte ne figure pas sur le diagramme de classes car elle n'est pas exprimable avec la

sémantique offerte par les diagrammes de classes. Cette contrainte permet de maximiser l'utilité des modèles d'interaction qu'un concepteur peut créer : tout modèle d'interaction exprime la manière d'interagir entre des dispositifs réels et virtuels, et des applications et réelles et virtuelles.

Le métamodèle présenté reflète le masquage dont bénéficie le concepteur. Il ne contient pas d'informations relatives aux types des données susceptibles d'être émises ou reçues par les instances. Le métamodèle est donc en réalité plus complet, pour permettre une configuration plus fine des interfaces. Un outil pour la spécification d'interface devrait permettre au concepteur de choisir le niveau de détail souhaité.

Ce métamodèle permet au concepteur de produire des modèles décrivant des interactions. Ces modèles contiennent des instances et des liaisons entre les ports de ces instances, certaines instances peuvent avoir une configuration. Ce métamodèle suppose l'existence de modèles à partir desquels les instances sont créées. La section qui suit présente le métamodèle qui permet à un programmeur de créer de tels modèles.

5.1.3 Métamodèles pour le programmeur

Le développeur est en charge de développer le code des composants configurables et le code des mandataires. Les mandataires sont des composants iPOJO qui leur permettent d'être pris en compte par une plateforme OSGi. Ces composants sont dynamiques [4] et sont plutôt simples à développer. Les composants configurables sont des médiateurs Cilia qui peuvent être instanciés au sein d'une chaine de médiation. Les médiateurs sont également dynamiques et plutôt simples à développer. Une fois les composants iPOJO et les médiateurs Cilia développés, DynaMo à besoin d'informations complémentaires pour les utiliser au sein d'une interface multimodale [5]. Le programmeur dispose donc de deux métamodèles : un métamodèle pour les mandataires et un métamodèle pour les composants configurables. Pour simplifier la présentation de métamodèles, les types de donnée qui constituent une partie commune entre ces deux métamodèles est d'abord présentée.

Pour travailler, le programmeur doit manipuler des types. Le but des types est de caractériser les ports selon la nature des données qu'ils peuvent traiter. Dix types ont été définis, ils sont représentés sur la figure 5.5. Le type **évènement** permet de caractériser le paradigme de l'appui sur un bouton : aucune valeur ne transite, le fait qu'il ait été actionné est la seule information transmise. Le type **chaine** permet de caractériser un port qui traite des chaines de caractères. Le type **entier** permet de caractériser un port qui traite des nombres entiers au sein d'un intervalle, l'intervalle est défini par sa borne inférieure et sa borne supérieure. Le type **entier générique** permet de caractériser un port qui traite des nombres entiers au sein de n'importe quel intervalle. Les **types flottant** et **flottant générique** sont similaires pour les nombres flottants. Le type **sélection** permet de caractériser un port qui traite des chaines parmi un ensemble de chaines possibles, défini par une énumération de ces chaines de caractères. Le type **sélection générique** permet de caractériser un port qui traite des chaines parmi n'importe quel ensemble de chaines possibles. Le type **liste** permet de caractériser un port qui traite plusieurs données en même temps. Les données qu'il traite sont ordonnées, et sont chacune caractérisées par un type. Enfin, le type **autre** permet de caractériser un port qui ne peut être caractérisé par aucun type déjà

4. Dynamiques dans le sens où leur cycle de vie (création, démarrage, arrêt, destruction) à l'exécution n'est pas lié au cycle de vie (démarrage, arrêt) de la plateforme.

5. Pour simplifier le développement de DynaMo, ces informations sont décrites dans un fichier à part. Idéalement, elles devraient être plus intégrées : soit décrites directement au sein du code des composants avec des annotations par exemple, soit ajoutées dans les fichiers de descriptions qui accompagnent les composants iPOJO et médiateurs Cilia.

décrit. Il est défini par un nom qui est une chaine de caractère. Un port d'entrée caractérisé par le type *autre* défini par un certain nom doit interpréter de manière correcte les données issues d'un port de sortie caractérisé par le type *autre* défini par le même nom.

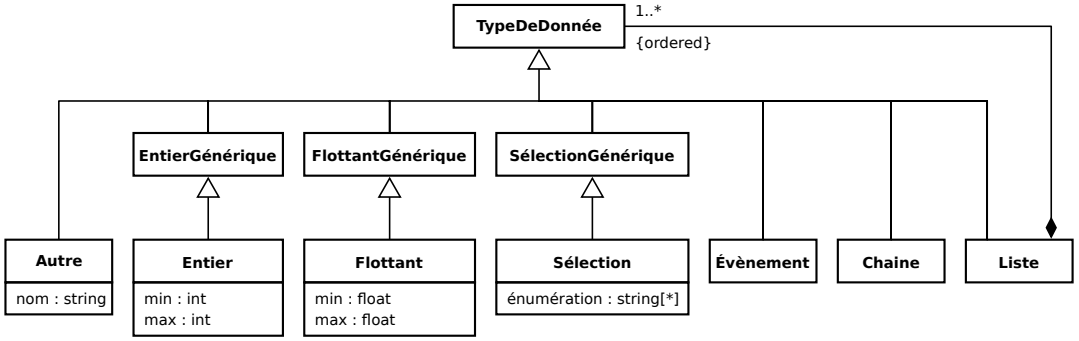


FIGURE 5.5 – Métamodèle des types de données.

Dans la suite, des ensembles de types peuvent être modélisés. Pour alléger la notation, des ensembles de types préexistent. Ainsi, l'ensemble « tous » correspond à l'ensemble *{Autre, EntierGénérique, FlottantGénérique, SélectionGénérique, Évènement, Chaine, Liste}*.

Les types venant d'être présentés, le métamodèle des mandataires, puis le métamodèle des médiateurs peuvent être présentés.

5.1.3.1 Métamodèle des mandataires

En plus de disposer du composant iPOJO implémentant le mandataire, DynaMo doit connaitre la manière de découvrir le dispositif ou l'application qu'il représente pour qu'une instance puisse être créée. DynaMo doit également connaître les ports et les caractéristiques des données qui transitent par ces ports. DynaMo doit aussi connaitre les groupements sémantiques des ports. Enfin, pour qu'un outil de spécification d'interface puisse aider un concepteur d'interface, des informations supplémentaires comme les noms, les descriptions et les représentations abordés dans la section 5.1.2 doivent être présentes. Ces dernières informations ne sont pas reprisent ici.

Le programmeur doit donc créer un modèle de mandataire qui regroupe toutes ces informations ; ce modèle doit être conforme à un métamodèle. Ce métamodèle est représenté sur la figure 5.6.

La manière de découvrir le dispositif ou l'application est résumée par le protocole de découverte et un discriminant. Le protocole de découverte est utilisé par le module de découverte pour connaitre quel sous-module prend en charge le dispositif ou l'application. Le discriminant est une chaine de caractères qui sert à configurer le sous-module : le sous-module peut alors, parmi toutes les entités qu'il reconnait, déterminer celles qui disposent d'un mandataire instanciable. La syntaxe de ce discriminant dépend du sous-module de recherche concerné. Par exemple, pour chaque entité que le sous-module Bluetooth découvre, il recherche pour chaque discriminant s'il constitue une sous-chaine du *friendly name* de l'entité. L'utilisation du discriminant est libre, il peut être interprété comme une expression rationnelle [6], un filtre LDAP, etc. DynaMo n'effectue pas de vérification au chargement du modèle, c'est le sous-module de découverte concerné qui est en charge de vérifier la conformité de la chaine selon le type d'expression qu'il attend.

6. Également appelé les expressions régulières.

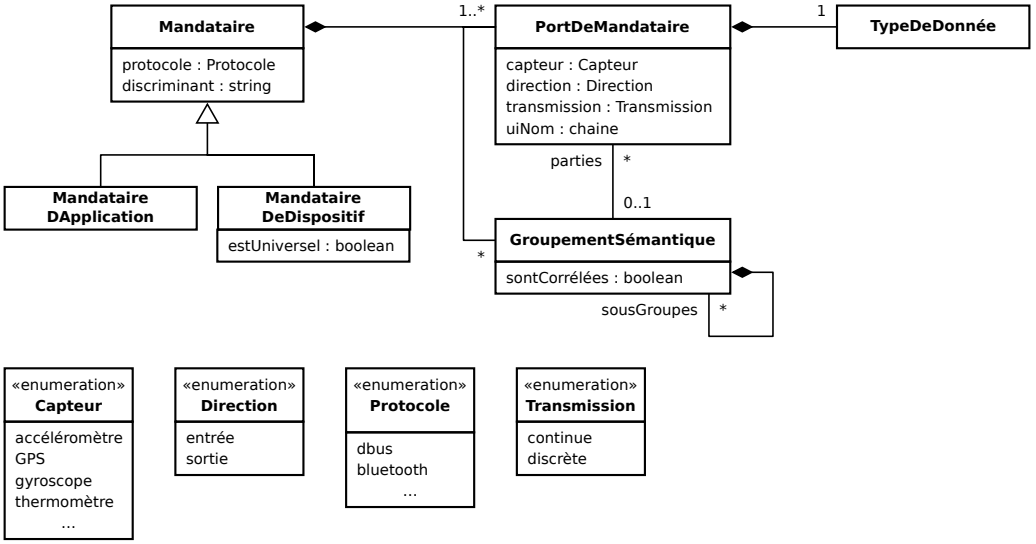


FIGURE 5.6 – Métamodèle des mandataires. La classe *TypeDeDonnées* est représentée dans la figure 5.5.

Le programmeur décrit un mandataire de dispositif ou d'application. Un mandataire de dispositif peut être universel si le dispositif qu'il représente répond aux critères de ce type de dispositif[7].

Un mandataire dispose d'au moins un port : un mandataire sans port ne peut communiquer avec l'extérieur. Un port est défini par la direction des données, le type de capteur dont sont issues les données, le type de transmission, un nom compréhensible par l'utilisateur et un type de donnée.

Les données peuvent provenir de l'extérieur et entrer dans le mandataire via le port, il est alors un port d'entrée : il reçoit des données d'un autre port. Les données peuvent sortir du mandataire pour être envoyées vers d'autres ports, il est alors un port de sortie.

La transmission peut être continue ou discrète. Cette information permet d'éviter de relier un capteur qui envoie un flux de données continu à une tâche qui fonctionne en mode discret. Un composant capable de sélectionner une valeur parmi le flux de données peut être inséré. En disposant de la fréquence d'envoie des données, il pourrait être possible de réaliser l'inverse, mais ce cas n'a pas été rencontré.

Le type de capteur est optionnel, il permet d'ajouter de la sémantique aux ports, pour éviter qu'un port d'entrée ne reçoive des informations qu'il interprèterait incorrectement. Par exemple, fournir des coordonnées GPS à un port qui attend des données issues d'un accéléromètre ne produit pas une interface utile. Le type de capteur définit précisément les données : le capteur GPS indique des données au format WGS 84, le gyroscope indique des données en radian, etc. Le nom compréhensible par l'utilisateur d'un port d'entrée d'une application est fourni à un dispositif universel comme expliqué dans la section 4.5.3.5 page 83.

Contrairement au type de capteur, le type de donnée a un rôle plus syntaxique : il reflète plus le type au sens langage de programmation. Ainsi, le type *autre* doit logiquement être défini par un type manipulable par le langage de programmation utilisé. Comme le langage Java est

7. Voir la section 4.5.3.5 page 83.

utilisé, le nom devrait être l'identifiant d'un type comme le nom canonique d'une classe [8]. Les trois types génériques ne sont à priori pas utilisés.

Enfin, un mandataire peut posséder des groupements sémantiques de ses ports tels qu'expliqués dans la section 4.5.3.3 page 81.

Il est concevable d'ajouter des informations concernant la nature relative ou absolue des données. Intuitivement, une souris envoie des coordonnées relatives, une surface tactile envoie des coordonnées absolues ; une tâche de réglage de volume sonore utilise une donnée relative x si elle a la sémantique « augmenter le volume de x », elle utilise une donnée absolue si elle a la sémantique « positionner le volume à x ». Plus formellement, une donnée est relative si sa prise en compte par une tâche dépend de l'état de cette tâche. Ainsi, considérer la nature relative ou absolue des données est complexe. En effet, créer automatiquement une conversion entre une donnée relative et une donnée absolue nécessite l'utilisation d'un médiateur capable de connaitre l'état d'une tâche et la manière de prendre en compte la donnée relative.

Toutes les informations que doit saisir le programmeur pour qu'un mandataire soit pris en compte par DynaMo viennent d'être présentées. La section suivante présente toutes les informations à saisir pour les médiateurs.

5.1.3.2 Métamodèle des médiateurs

Les médiateurs Cilia constituent les briques de base des chaines de médiation. Du point de vue du concepteur d'interaction, les mandataires prennent le terme de « composant configurable ». Comme les mandataires, un modèle doit être créé pour qu'un médiateur puisse être exploité par DynaMo. Ce modèle doit être conforme à un métamodèle représenté sur la figure 5.7. Un modèle contient des informations à propos de la configuration, des ports et des types acceptables par les ports. Pour qu'un outil de spécification d'interface puisse aider un concepteur d'interface, des informations supplémentaires comme les noms, les descriptions et les représentations abordées dans la section 5.1.2 doivent être présentes. Ces dernières informations ne sont pas reprisent ici.

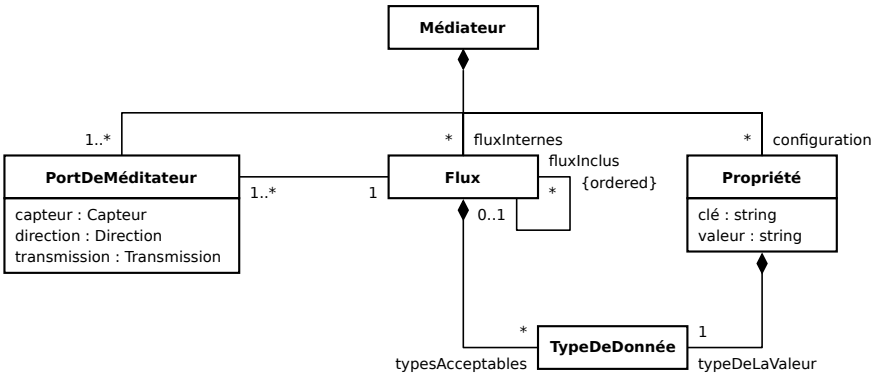


FIGURE 5.7 – Métamodèle des médiateurs. La classe *TypeDeDonnées* est définie dans la figure 5.5, les types *Capteurs*, *Direction* et *Transmission* sont définis dans la figure 5.6.

La configuration d'un médiateur vise à permettre sa spécialisation par un concepteur d'interaction. La configuration consiste en un ensemble de propriétés. Une propriété est composée d'une clé et d'une valeur. L'information de type est uniquement présente pour être exploitée par un outil de spécification d'interface, DynaMo ne l'utilise pas et ne la vérifie pas lors du chargement

8. Voir la section 6.7 *Fully Qualified Names and Canonical Names* de [Gosling *et al.*, 2005, p. 145].

du modèle en mémoire. En effet, la configuration est fournie à une instance de médiateur lorsque celle-ci est créée. C'est au médiateur de vérifier la conformité de la valeur. Les propriétés n'étant pas imposées, le programmeur est libre de définir toute propriété qui peut être utile pour son médiateur. Comme le programmeur réalise le code du médiateur, celui-ci peut interpréter les propriétés comme bon lui semble. Toute propriété peut être redéfinie par le concepteur d'interaction. La valeur d'une propriété dans un modèle de médiateur devient la valeur par défaut dans le métamodèle de conception.

Les attributs des ports des médiateurs sont semblables aux attributs des ports des mandataires. La manière de typer les ports est différente : alors que les ports des mandataires sont typés directement, les ports des médiateurs sont typés via des flux. Cette particularité est due à la conjugaison de trois raisons. Tout d'abord, les types sont volontairement masqués au concepteur d'interaction : il manipule des composants qui implémentent des fonctions. Donc, les médiateurs doivent être génériques vis-à-vis des types : un médiateur peut accepter différents types sur un port, les types que doivent traiter les ports sont connus par une instance lors de sa création. Enfin, le gestionnaire autonomique doit pouvoir inférer les types de données pour éviter de relier des ports incompatibles et insérer des conversions de type entre les médiateurs. Les deux premières raisons impliquent que plusieurs types peuvent être associés à un port. La dernière raison implique l'existence d'un mécanisme pour inférer le type des données qui transite par un port. Ce mécanisme repose sur le concept de flux interne. Un flux interne est associé à un ou plusieurs ports : deux ports ayant le même flux associé impliquent que les données qui transitent par l'un ont le même type que les données qui transitent par l'autre. L'étape qui consiste à sélectionner le type des données qui transitent par un port s'appelle la résolution des types. Ce type de donnée est sélectionné parmi les types de donnée acceptables par le port.

Deux exemples permettent de mettre en lumière le fonctionnement des flux. Le premier concerne un médiateur « déclencheur », le deuxième concerne un médiateur « fusion ».

Un médiateur « déclencheur » contrôle l'envoi de données. Ce médiateur peut être utilisé pour transformer une transmission continue en transmission discrète. Plus formellement, il envoie sur le port « sortie » la dernière valeur reçue sur le port d'entrée « donnée » à chaque fois qu'il reçoit une donnée sur un port « déclenchement ». Ce médiateur est générique puisque le traitement des données n'est pas dépendant de leur type, mais le type des données qui transitent par le port de sortie sont de même type que les données qui transitent par le port « donnée ». Le gestionnaire autonomique doit connaitre cette information pour ne pas relier de ports incompatibles : il doit connaitre les « flux internes » des données qui lui permettent d'inférer les types de données qui transitent par les ports. Le modèle de ce médiateur est représenté sur la figure 5.8.

Un médiateur « fusion » permet de mettre des données ensemble. Plus formellement, il envoie sur le port « sortie » une liste composée des données reçues sur ses ports « entréeA » et « entréeB ». Le modèle de ce médiateur est représenté sur la figure 5.9. Ce médiateur est générique car il peut fusionner tout type de donnée. Cependant, les types des éléments au sein de la liste de sortie ne sont pas aléatoires : ils sont égaux aux types qui transitent dans les ports d'entrées. Cette relation de type entre des ports est modélisée par l'association *flux inclus*. Cette relation est ordonnée, elle reflète l'ordre des types dans la liste.

Les modèles que doit produire le programmeur viennent d'être présentés. À partir de ces modèles, un outil de spécification d'interface peut présenter des applications, dispositifs et composants configurables à un concepteur d'interaction dans le but d'être assemblés. Les informations contenues dans ces modèles se retrouvent également dans les modèles qu'utilise le gestionnaire autonomique pour générer une interface. Ces modèles sont l'objet de la section suivante.

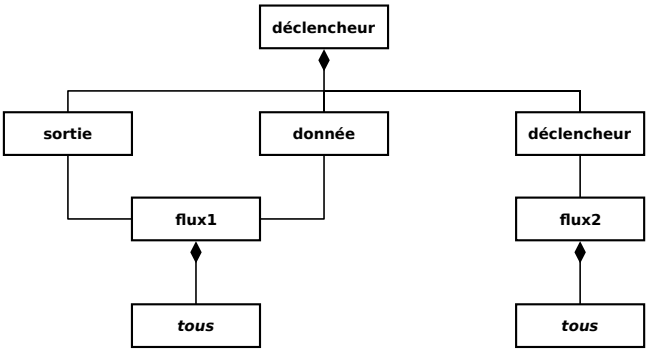


FIGURE 5.8 – Exemple de modèle de médiateur : le médiateur *déclencheur* dispose de trois ports et de deux flux. Les ports *sortie* et *donnée* sont traversés par le même flux *flux1*, le port *déclencheur* est traversé par un autre flux. Les deux flux acceptent tous les types de données, ce qui est résumé par l'instance *tous*. Au cours de la résolution des types, les types qui transitent par les ports *sortie* et *donnée* seront nécessairement résolus par le même type.

5.1.4 Métamodèle pour le gestionnaire autonomique

Pour réaliser la transformation entre des modèles de conception créés par un concepteur d'interaction et un modèle de chaine de médiation, le gestionnaire autonomique exploite des modèles contenant les mêmes informations que les modèles de conception mais avec une structuration différente. Cette différente structuration permet au gestionnaire autonomique de traiter les modèles issus du langage de conception tel que nous les avons définis, mais également beaucoup d'autres modèles qui ne sont pas concevables avec le langage de conception. Dit autrement, l'ensemble des modèles conformes au métamodèle de conception est un sous-ensemble des modèles conformes au métamodèle pour le gestionnaire autonomique. Il est représenté complètement dans l'annexe A page 131.

Ce métamodèle pour le gestionnaire autonomique est présenté en deux parties. La première partie présente la partie « composant » du métamodèle, la deuxième partie présente la partie « interaction » qui s'appuie sur la première partie.

5.1.4.1 Métamodèle des composants

Lors de l'exécution, une chaine de médiation est composée d'instances de médiateurs et d'instances de mandataire. Cette différence technologique n'est pas importante pour le gestionnaire autonomique. Ainsi, le gestionnaire autonomique se préoccupe uniquement de composants, qui peuvent être des dispositifs – éventuellement universels, et des applications. Le métamodèle des composants est représenté sur la figure 5.10. La classe est appelée « composant » pour simplifier le discours, mais il s'agit bien d'instances de composants.

Les modèles de médiateurs et de mandataires doivent pouvoir être représentés avec ce modèle sans perte d'information. Ainsi, les ports ne sont plus typés directement comme c'est le cas des mandataires, mais ils sont typés avec des flux. Les ports d'un mandataire deviennent alors les ports d'un composant qui ont chacun un flux différent. Chaque flux contient un unique type acceptable qui est égal au type du port correspondant. Les ports des médiateurs n'ont pas besoin d'une telle transformation.

La définition des groupements sémantiques ne change pas. Les groupements sont optionnels pour les mandataires, ils n'existent pas pour les médiateurs. Ils sont alors optionnels pour les composants.

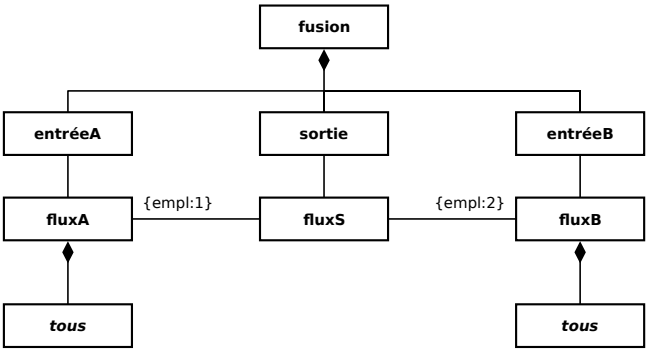


FIGURE 5.9 – Exemple de modèle de médiateur : le médiateur *fusion* dispose de trois ports et de trois flux. Les ports *entréeA*, *entréeB* et *sortie* sont traversés chacun par un flux distinct, mais le flux de *sortie* est composé du flux du port *entréeA* et du flux du port *entréeB*. Deux flux acceptent tous les types de données qui est résumé par l'instance *tous*. L'association *fluxInclus* est ordonnée : *fluxA* est à l'emplacement 1, *fluxB* est à l'emplacement 2. Les données qui sortent sont donc des listes dont le premier élément est du type de *fluxA* et dont le deuxième élément est du type de *fluxB*.

Les attributs des ports sont l'union des attributs des ports des mandataires et des médiateurs. Les attributs « capteur » et « uiNom » sont optionnels. Leur sémantique ne change pas.

La configuration d'un composant est fournie à l'instance exécutable lors de son démarrage. Une configuration est l'unique moyen que possède le gestionnaire autonomique pour fournir de l'information aux mandataires et médiateurs exécutables. La configuration est constituée des propriétés des médiateurs : les clés sont les mêmes, les valeurs sont celles fournies par le modèle de conception. Si aucune valeur n'est fournie par le modèle de conception, c'est la valeur fournie par le médiateur qui est prise. Le gestionnaire autonomique ajoute certaines propriétés prédéfinies. Les médiateurs génériques peuvent avoir besoin des types qui résolvent leurs ports. Cette information leur permet de se configurer lors de leur démarrage, et leur évite d'instrospecter les données qu'ils ont à traiter. Une propriété est ajoutée aux composants de médiateurs pour qu'ils sachent comment se lier à la chaine de médiation.

Enfin, un dispositif ou une application virtuels sont modélisés par un seul composant qui a le même nombre de ports d'entrée et de sortie. Chaque couple de ports est lié à un flux.

Mis à part les propriétés qui sont étendues pour des raisons pratiques, les autres concepts sont identiques à ceux présentés jusqu'à maintenant. Suit la deuxième partie qui présente les fragments d'interaction, et ajoute le concept de dépendance qui ne fait pas partie des concepts des métamodèles présentés jusqu'à maintenant.

5.1.4.2 Métamodèle des fragments d'interaction

Le métamodèle de conception permet de spécifier des interactions mettant en jeu des dispositifs et des applications – éventuellement virtuels – en créant des liaisons et des instances de composant configurables. L'utilisation de dispositifs et applications virtuels donnent l'illusion au concepteur de concevoir des interactions complètes. Le gestionnaire autonomique considère ces modèles d'interaction comme des fragments d'interaction à assembler, qui ont des dépendances entre eux. Ainsi, une spécification d'interaction entre un dispositif et une application devient un fragment d'interaction qui dépend du dispositif et de l'application. Le dispositif et l'application

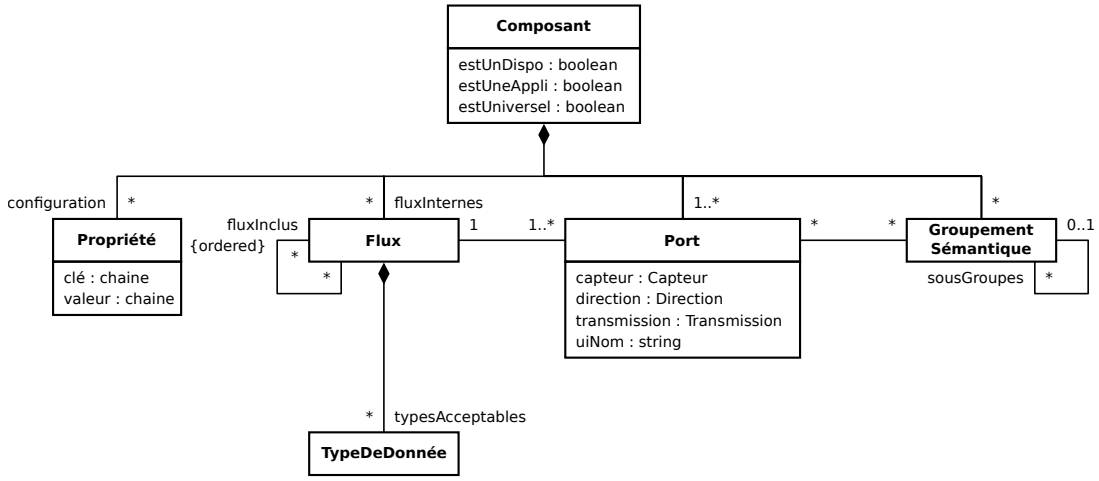


FIGURE 5.10 – Métamodèle des composants pour le gestionnaire autonomique. La classe *TypeDeDonnées* est définie dans la figure 5.5, les types *Capteurs*, *Direction* et *Transmission* sont définis dans la figure 5.6.

sont alors considérés comme des fragments d'interaction. Le métamodèle des fragments d'interaction est représenté sur la figure 5.11. Ce métamodèle permet au gestionnaire autonomique de réaliser l'étape de sélection de modèles présentée dans la section 4.5.2 page 71.

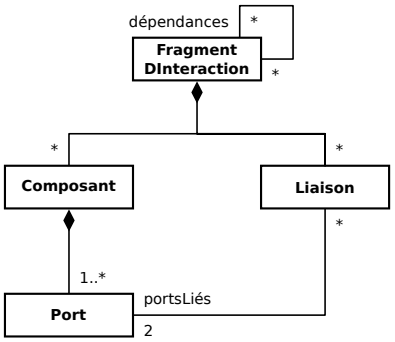


FIGURE 5.11 – Métamodèle des fragments d'interaction pour le gestionnaire autonomique. Les classes *Composant* et *Port* sont définies dans la figure 5.10.

Un fragment d'interaction est composé d'instances de composant et de liaisons. Les liaisons ne peuvent pas lier n'importe quels ports : une liaison appartenant à un fragment d'interaction f ne peut lier que les ports des composants appartenant à f et les ports des composants appartenant à un fragment d'interaction qui est une dépendance directe de f.

Une contrainte existe sur les dépendances : les dépendances ne doivent pas constituer de cycle [9]. Le métamodèle de conception garantit l'absence de cycle dans les dépendances, car les dispositifs et les applications n'ont aucune dépendance. La création de fragment d'interaction sans passer par le langage de conception nécessite de tenir compte de cette contrainte.

9. Exemple de cycle entre deux fragments f_1 et f_2 : $f_1.dépendances = \{f_2\}$ et $f_2.dépendances = \{f_1\}$.

Les modèles conformes à ce métamodèle sont soit écrits directement, soit le résultat d'une conversion des modèles de conception et des modèles des mandataires et médiateurs. La conversion des modèles des mandataires et des médiateurs en modèles de composant a été abordée dans la section précédente : elle est simple et systématique. La conversion d'un modèle de conception est systématique et légèrement plus complexe.

Le but de cette transformation est de remplacer les applications et les dispositifs dans un modèle de conception par des dépendances. Tout d'abord, pour chaque application ou dispositif est créé un fragment d'interaction qui le contient. Ainsi, les applications et dispositifs peuvent constituer des dépendances de fragment d'interaction. Enfin, pour chaque modèle de conception, tout dispositif et application – virtuel ou non – est remplacé par la dépendance vers le fragment d'interaction qui lui correspond. Dans le modèle de conception ne reste plus que les composants configurables et les liaisons.

5.1.5 Synthèse

Cette section a présenté trois métamodèles. Chaque métamodèle est destiné à un acteur : ils sont dédiés à la tâche qu'ils permettent de faire. Le métamodèle pour le concepteur d'interaction est dédié à la création d'interfaces multimodales. Le métamodèle pour le programmeur est dédié à la description de mandataire et de médiateur. Le métamodèle pour le gestionnaire autonomique est dédié à faciliter l'écriture des algorithmes présentés dans la proposition.

DynaMo possède un module de chargement des modèles. Ce module parse des fichiers écrits en XML et les convertit pour obtenir une représentation en mémoire adéquate. Ces fichiers sont dans une arborescence spécifique. Cette arborescence constitue un dépôt. DynaMo est capable de gérer plusieurs dépôts, locaux ou distants. Si un dépôt est local, alors DynaMo est capable de détecter les changements et met à jour l'interface si besoin.

5.2 Plateforme d'intégration

La plateforme d'intégration est en fait composée de deux plateformes. La plateforme à service qui permet de masquer l'hétérogénéité des technologies des communications avec l'environnement au reste de la plateforme. La plateforme de médiation crée des chaines de médiation qui constituent les canaux de communication que suivent les données entre les dispositifs et les applications.

Cette section aborde la plateforme à service, puis la plateforme de médiation.

5.2.1 Plateforme à service

La plateforme à service est composée d'un module de découverte, de mandataires et d'un registre. La plateforme à service ne différencie pas les dispositifs et les applications, elles sont donc désignées par « entités » dans la suite de cette section.

Le module de découverte est chargé de détecter l'apparition des entités présentes dans l'environnement. Lors de la découverte d'une entité, il instancie un mandataire qui est chargé d'adapter la communication entre cette entité et la chaine de médiation. Comme ce mandataire est un service, aussitôt créé, il s'enregistre dans le registre de la plateforme.

Le module de découverte est également chargé de détecter les disparitions des entités de l'environnement. Lors de la détection d'une disparition, il détruit le mandataire correspondant. Le mandataire étant un service, il se désenregistre du registre juste avant d'être détruit.

5.2.1.1 Module de découverte

Le module de découverte est constitué d'un ensemble de sous-modules. Chaque sous-module est implémenté par un service OSGi ou un composant iPOJO. Ils proposent tous le même service. Avec OSGi comme avec iPOJO, la spécification d'un service correspond à l'interface qui est implémenté par la classe qui implémente le service. Ainsi, les services de découverte implémentent tous l'interface *DynamoDiscovery* qui est fournie par DynaMo. Cette interface Java est constituée de trois méthodes qui permettent de récupérer le nom du sous-module, de demander le suivi d'une entité, et d'arrêter le suivi d'une entité. La figure 5.12 représente le module de découverte et trois sous-modules.

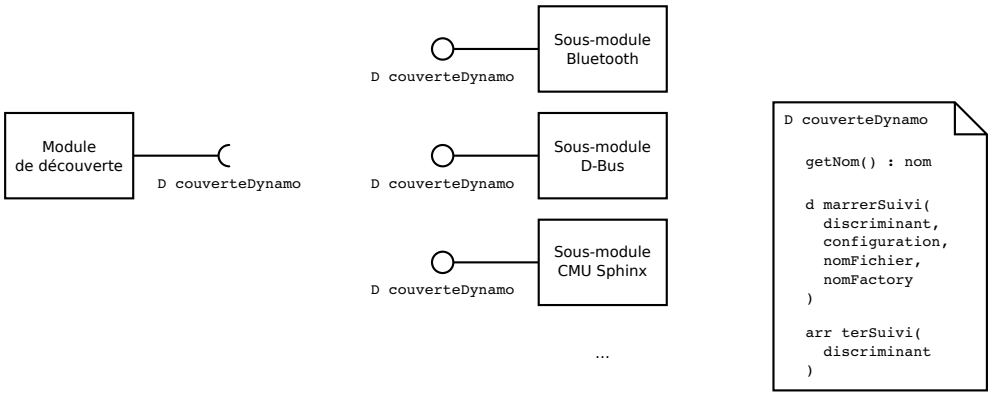


FIGURE 5.12 – Module de découverte et sous-modules : les sous-modules sont implémentés sous la forme de services qui implémentent la spécification *DécouverteDynamo*.

Le nom d'un sous-module correspond au protocole qui est géré par le service. Ce nom permet à DynaMo de trouver le bon sous-module lorsqu'il veut configurer la découverte d'une entité. DynaMo utilise l'attribut *protocole* d'un modèle de mandataire [10].

La méthode pour demander de suivre une entité comporte quatre paramètres : un discriminant fourni par le modèle du mandataire, le nom de fichier du bundle dans lequel se trouve le code du mandataire, le nom de la *factory* du composant iPOJO qui permet de créer un mandataire et enfin une configuration à fournir à l'instance du mandataire lors de sa création. Le nom du fichier et le nom de la factory sont également présents dans le modèle du mandataire mais ont été omis dans la présentation du métamodèle des mandataires.

Classiquement, un service de découverte utilise un protocole de découverte pour trouver des entités. Il peut donc découvrir des entités qui n'intéressent pas DynaMo. Le discriminant est une chaine de caractères qui lui permet de savoir si une entité découverte est utile pour DynaMo : cette chaine discrimine les entités. La manière d'utiliser le discriminant est propre à chaque service. En effet, les protocoles de découverte n'utilisent pas forcément la même manière d'identifier les entités qu'ils permettent de découvrir. Par exemple, le protocole Bluetooth fournit un *friendly name* qui caractérise une entité. Le protocole D-Bus offre un identifiant unique pour chaque bus. Cet identifiant est la concaténation du nom de l'entité et d'un numéro unique. Le service de découverte D-Bus recherche alors parmi tous les bus disponibles ceux dont l'identifiant commence par le discriminant.

Un mandataire est un composant iPOJO. Un composant iPOJO est compatible avec OSGi. Le composant est donc fourni sous la forme d'un bundle, qui existe sous la forme d'un fichier.

10. Voir figure 5.6 de la section 5.1.3.1 page 94.

Un bundle doit être installé et démarré dans l'environnement OSGi pour que les services qu'il fournit soient exploités [11]. Pour éviter d'installer tous les bundles dans l'environnement OSGi, DynaMo fournit un service de gestion des bundles que les services de découverte doivent utiliser pour installer et démarrer les bundles dont ils ont besoin. Le nom de fichier du bundle est fourni à ce service de gestion lorsqu'une entité est découverte.

Un service de découverte n'instancie pas lui même les composants iPOJO. Il délègue cette tâche à un service qui fait partie de l'outil ROSE [12]. ROSE a besoin du nom de la factory iPOJO pour créer l'instance. ROSE peut fournir une configuration à une instance qu'il crée. Ainsi, ces deux informations sont fournies à ROSE lorsque le service reconnait une entité dans l'environnement. Deux raisons ont poussé cette délégation. Premièrement, ROSE masque certains détails concernant iPOJO au programmeur qui développe un service de découverte. Ensuite, ROSE offre la possibilité de faire communiquer plusieurs plateformes OSGi : il permet de faire de l'informatique distribuée [13]. L'informatique distribuée a du sens au sein d'une interaction avec un environnement pervasif car les dispositifs et les applications ne sont pas forcément tous reliés à un ordinateur central.

Lorsqu'un service de découverte détecte la disparition d'une entité, il demande à ROSE de détruire l'instance. Découvrir la disparition d'une entité n'est pas forcément une capacité des protocoles de découverte. Le protocole Bluetooth par exemple permet de découvrir l'arrivée d'une entité, mais pas sa disparition. Seul le code qui possède effectivement une connexion avec un dispositif est prévenu de la perte de cette connexion. C'est le cas du mandataire. Pour pallier ces cas, un service de découverte ajoute à la configuration un identifiant unique de l'entité. Ainsi, lorsque le mandataire détecte la disparition de l'entité, il prévient le service de découverte, qui se comporte alors comme s'il avait détecté lui même la disparition de l'entité [14].

La méthode pour arrêter de suivre une entité comporte un seul paramètre : le discriminant. Cette méthode doit logiquement demander la destruction des instances concernées.

Puisque les sous-modules sont implémentés par des services similaires, DynaMo les découvre lors de l'exécution, il n'a pas besoin de les connaitre à l'avance. Ce fonctionnement permet à un développeur de créer un sous-module de découverte sans modifier DynaMo. Ainsi, pour que DynaMo puisse prendre en compte un dispositif, soit il existe déjà un sous-module capable de le reconnaitre, alors il suffit de créer le mandataire et son modèle, soit un tel sous-module n'existe pas, alors le développeur doit créer en plus un service qui permet de découvrir l'entité.

Les services de découverte sont présentés comme liés à un protocole de découverte. Ce n'est pas toujours le cas. En particulier, la reconnaissance de la parole et la synthèse vocale ne s'appuient pas sur des protocoles de découverte. Ces deux systèmes ajoutés dans DynaMo ne reposent pas sur la découverte de dispositifs réels, mais sur la présence de ces systèmes. La synthèse vocale est réalisée avec FreeTTS [15]. Diverses raisons peuvent amener le système à ne pas fonctionner, notamment l'absence de ressource audio [16] ou son utilisation exclusive par un autre système. Tester toutes les conditions de fonctionnement demande d'extraire du code les sources d'erreurs possibles. Pour simplifier, la « découverte » de FreeTTS consiste à essayer de lancer le système. Si cet essai est concluant, alors le sous-module dédié à FreeTTS demande la création de l'instance du mandataire. La détection de l'arrêt du système ne peut pas se faire simplement : essayer de

11. Voir la figure 3.3 de la section 3.1.2 page 40.
12. Site officiel : `http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Rose`.
13. Cette possibilité n'a pas été testée.
14. Il existe une communication asynchrone entre la notification du service de découverte par le mandataire et sa destruction : la notification n'est pas bloquante car elle empêcherait la destruction du mandataire qui résulterait par un interblocage (*deadlock*).
15. Site officiel : `http://freetts.sourceforge.net/`.
16. Traduction approximative de *line*.

lancer le système alors que le mandataire l'a déjà lancé échoue systématiquement. La détection de la « disparition » de FreeTTS est donc similaire à Bluetooth : c'est le mandataire lui-même qui notifie le service de découverte. La reconnaissance de la parole est réalisée avec CMU Sphinx [17]. Son service de découverte est similaire. Ainsi, dans certain cas, un service de découverte peut être restreint à la découverte d'une seule entité.

Le module de découverte permet donc de créer et détruire des instances de mandataire. Entre sa création et sa destruction, un mandataire peut être amené à communiquer avec une chaine de médiation. La section suivante présente cette communication.

5.2.1.2 Communication avec une chaine de médiation

Le but d'un mandataire est d'adapter la communication entre une chaine de médiation et l'entité qu'il représente. La communication entre le mandataire et le dispositif dépend du dispositif et DynaMo ne peut pas faciliter cette partie. La communication entre le mandataire et la chaine de médiation a été simplifiée au maximum.

Un port d'entrée prend la forme d'une méthode, un port de sortie prend la forme d'une variable. Ainsi, à chaque donnée qui est reçue sur un port d'entrée, la méthode correspondante est appelée, la valeur est fournie dans un paramètre de la méthode. À chaque fois qu'une donnée doit être émise sur un port de sortie, la variable correspondante doit être affectée par la valeur à émettre. La correspondance entre les ports et les méthodes (ou les variables) est donnée dans la description du mandataire : à chaque port est ajouté le nom de la variable ou de la méthode.

Ce mécanisme permet au code du mandataire de n'avoir aucune dépendance sur DynaMo. Il suffit d'ajouter un *handler* iPOJO au composant. Ajouter un handler se fait en insérant une ligne dans la description du composant. Ce handler est fourni par DynaMo, il est générique. Il s'adapte au composant auquel il est attaché en utilisant la configuration qui lui est passée lors de son instanciation. Dans la section précédente, cette configuration a été mentionnée : elle est passée en paramètre au service de découverte, qui la passe à ROSE. Cette configuration est créée par DynaMo à partir du modèle du mandataire.

Concrètement, c'est le handler qui communique avec la chaine de médiation et qui offre un service. Le handler est capable d'appeler des méthodes de son composant et de surveiller la modification des variables grâce aux mécanismes de liaison injectés par iPOJO dans le composant [18]. Lors de sa création, le service que propose le handler est automatiquement ajouté par iPOJO dans le registre OSGi. Lors de sa destruction, il est automatiquement retiré du registre.

Les mandataires sont écrits avec le langage de programmation Java. Les types de DynaMo correspondent aux types de Java. Ainsi, un port d'entrée caractérisé par le type *Chaine* correspond à une méthode dont le paramètre est de type *String* de Java. Le type *Autre* correspond à un paramètre de type *Object* de Java. Le type *Liste* correspond à un paramètre de type *List* contenant des valeurs de type *Object* [19]. Le type *Évènement* ne convoie aucune valeur, il correspond donc à l'absence de paramètre. Les types *Entier* et *Flottant* correspondent à des paramètres *Integer* ou *int* et *Float* ou *float* dont la valeur est nécessairement incluse dans l'intervalle qui les caractérise. Le type *Sélection* correspond à un paramètre de type *String* dont la valeur existe dans l'énumération qui le caractérise.

Dès que les mandataires sont instanciés une chaine de médiation peut être créée au sein de la plateforme de médiation. Cette plateforme de médiation est l'objet de la section suivante.

17. Site officiel : `http://cmusphinx.sourceforge.net/`.
18. Voir la section 3.1.3 page 41.
19. Le langage Java ne permet pas de créer des objets de type *List* contenant des objets de types différents.

5.2.2 Plateforme de médiation

La plateforme de médiation reçoit un modèle de chaine de médiation du gestionnaire autonomique et crée une chaine conforme au modèle. Cette chaine constitue l'interface multimodale entre les mandataires : d'une part les médiateurs effectuent des traitements sur les données, d'autre part les liaisons entre les médiateurs dirigent les données.

La plateforme de médiation est essentiellement constituée de Cilia [20]. Cilia propose une API pour construire et modifier des chaines lors de l'exécution. Ainsi, a été ajouté un module capable de convertir le modèle de chaine de médiation en une suite d'appels à cette API. Cette suite d'appels construit une chaine conforme au modèle.

Trois raisons font de Cilia un candidat idéal pour l'implémentation de la plateforme de médiation : les chaines de médiations peuvent être construites et détruites lors de l'exécution, les médiateurs sont configurables, et les médiateurs sont construits simplement à partir de trois briques de bases.

Les trois briques de base sont les planificateurs, les processeurs et les répartiteurs. Le programmeur peut créer des briques de base, et construire des médiateurs en les assemblant. DynaMo et Cilia fournissent quelques planificateurs, processeurs et répartiteurs.

5.2.3 Synthèse

Le module de découverte de DynaMo est extensible, les sous-modules s'apparentent à des greffons [21]. Un sous-module est implémenté par un service. Ce service doit implémenter une spécification simple qui contient trois méthodes. DynaMo met à disposition deux services qui facilitent l'écriture d'un service de découverte : un service de ROSE qui prend en charge les créations et destructions des mandataires, un service de gestion des bundles.

La communication entre un mandataire et une chaine de médiation est transparente du point de vue du mandataire : une méthode correspond à un port d'entrée, une variable correspond à un port de sortie. La communication entre un mandataire et une chaine de médiation est réalisée par un handler iPOJO. Ce mécanisme facilite énormément l'écriture du code du mandataire : il n'a aucune dépendance sur Dynamo. Le programmeur n'a pas besoin de connaitre ce mécanisme pour développer un mandataire.

La plateforme de médiation se base sur Cilia. Cilia fournit les propriétés nécessaires à la plateforme : les chaines peuvent être contrôlées lors de l'exécution, elles sont constituées de médiateurs configurables. Ces médiateurs sont construits par assemblage de trois briques de base.

Les technologies utilisées – iPOJO, ROSE et Cilia – ont toutes OSGi en commun. Implémenté ainsi, DynaMo est une plateforme cohérente qui profite du dynamisme et de la robustesse offerts par OSGi. OSGi n'est qu'un choix technologique, rien dans la proposition ne repose explicitement dessus.

La section suivante présente le gestionnaire autonomique, qui repose également sur OSGi pour être extensible.

20. Voir la section 3.1.4 page 42 qui présente Cilia.
21. *Plugins* en anglais.

5.3 Gestionnaire autonomique

Le gestionnaire autonomique est chargé de transformer des fragments d'interaction [22] en une interface concrète. Son travail se découpe en deux phases : la sélection des fragments d'interaction les plus adaptés à un environnement, puis la transformation de ces fragments en un modèle de chaine de médiation.

La sélection a été expliquée en détail dans la proposition [23], et nécessite peu d'explications complémentaires. La construction d'ensembles cohérents et valides correspond à l'algorithme 4.1 avec quelques optimisations. La sélection de modèles est l'application successive des heuristiques.

La suite de cette section explique la manière dont a été implémentée la transformation.

5.3.1 Transformation : solveurs

La transformation consiste à retirer et ajouter des composants et des liaisons, à compléter des configurations de composant et à transformer le modèle d'interaction en un modèle de chaine de médiation. Cette transformation est implémentée comme une suite de transformations.

La première transformation consiste à mettre tous les modèles des fragments d'interaction ensemble pour obtenir un unique modèle d'interaction. Ainsi, tous les composants et toutes les liaisons des fragments d'interaction sont mis dans un unique fragment d'interaction qui n'a pas de dépendance.

Les transformations suivantes concernent chacune un aspect : adaptation des types, adaptation des intervalles, utilisation des ports libres, etc. Ces simples transformations successives sont plus ou moins indépendantes. Un module logiciel qui réalise une transformation est appelé un solveur. Le gestionnaire autonomique appelle les solveurs successivement. À chaque solveur est passé le modèle d'interaction.

Comme pour les sous-modules de découverte, les solveurs sont implémentés sous la forme de services. Un solveur doit implémenter la spécification *Solveur* pour que DynaMo l'utilise. Le gestionnaire autonomique connait à l'avance quelques solveurs nécessaires. D'autres solveurs peuvent être ajoutés à DynaMo. Ce mécanisme permet d'étendre très simplement le gestionnaire autonomique sans modifier directement son code.

5.3.1.1 Solveurs par défaut

Quatre solveurs sont fournis avec DynaMo. Dans l'ordre d'application : le solveur de ports libres, l'adaptateur de type, l'adaptateur d'intervalle, et l'adaptateur de communication entre une chaine et les mandataires.

Le solveur de ports libres se comporte comme décrit dans la section 4.5.3.3 page 81. Ce solveur ajoute des connexions et éventuellement des composants de fusion.

L'adaptateur de type vérifie toutes les connexions et insère des composants d'adaptation de type si nécessaire.

L'adaptateur d'intervalle vérifie toutes les connexions et insère des composants d'adaptation d'intervalle si nécessaire.

L'adaptateur de communication entre une chaine et les mandataires configure les composants de mandataires pour que chacun se lie à l'instance du mandataire auquel il correspond.

22. Le métamodèle des fragments d'interaction est représenté sur la figure A.4 page 131.
23. Voir la section 4.5.2 page 71.

5.3.1.2 Solveur d'extension

Le gestionnaire autonomique peut être étendu avec d'autres solveurs. Il suffit que ces solveurs implémentent le service adéquat. Les solveurs d'extension sont exécutés après les solveurs par défaut. L'ordre d'exécution des solveurs d'extension n'est pas garanti.

Les solveurs d'extension ont les mêmes possibilités que les solveurs par défaut. À la différence des solveurs par défaut, les solveurs d'extension ne sont pas connus à l'avance par le gestionnaire autonomique. Étendre le gestionnaire autonomique avec un solveur ne nécessite pas de modifier le gestionnaire autonomique.

Deux exemples de solveur sont maintenant présentés : un traceur, et un solveur de mesure de confiance.

Un solveur de trace constitue un outil pour analyser les données qui circulent au sein d'une chaine de médiation pendant l'exécution. Il consiste à capturer les données qui transitent par un port donné. Pour que le solveur prenne en compte un port, l'instance de médiateur de ce port doit être configurée avec deux propriétés. Une propriété contient la liste des ports à considérer, l'autre propriété configure la manière d'utiliser les données capturées : enregistrement dans un fichier, affichage à l'écran sous forme de courbe graphique, affichage sur la sortie standard, etc. Lorsque le solveur est exécuté par le gestionnaire autonomique, il recherche tous les composants qui possèdent la propriété contenant la liste de ports, et intercale une instance de composants par port à considérer.

Les techniques de reconnaissance, comme la reconnaissance de la parole ou de gestes, sont imparfaites. Un système de reconnaissance peut alors fournir un nombre qui reflète la probabilité d'avoir effectué une reconnaissance correcte. Ce nombre est souvent appelé mesure de confiance. Si toutes les mesures de confiance sont exprimées sur la même échelle, un pourcentage par exemple, il devient possible pour un utilisateur de régler de manière globale la finesse des systèmes de reconnaissance utilisés en spécifiant une valeur de seuil en-dessous de laquelle l'interface doit ignorer ce qui a été reconnu. Le solveur de mesure de confiance ajoute les mécanismes nécessaires à une interface pour qu'un tel seuil puisse être pris en compte. Ainsi, pour chaque port offrant une mesure de confiance, il doit connaitre le second port auquel il est appliqué. Le solveur insère alors un médiateur qui possède deux entrées et une sortie. La première entrée récupère la mesure de confiance et la compare au seuil configuré par l'utilisateur final. Si la mesure est plus grande que le seuil, alors le médiateur recopie sur la sortie la valeur reçue sur le second port d'entrée qui correspond au résultat de la reconnaissance. Un port qui fournit une mesure de confiance est toujours libre avant l'exécution de ce solveur : il déclare un type de capteur qui n'est utilisé par aucun port d'entrée [24].

D'autres solveurs d'extension peuvent être définis. Ces extensions s'appuient en général sur la possibilité d'ajouter des propriétés aux instances de médiateur. Ces propriétés ne sont pas nécessairement connues à l'avance par DynaMo.

24. Voir la section 5.1.3.1 page 93.

5.4 Conclusion

5.4.1 Quelques chiffres

Sonar [25] a été utilisé pour superviser diverses métriques de DynaMo. Voici quelques chiffres pour donner une idée de son ampleur. DynaMo représente en tout 209 fichiers Java qui contiennent 26 400 lignes, dont 13 300 lignes de code et 22.7% de commentaires. Le code dupliqué représente 149 lignes (0.6%). Il est composé de 259 classes, 1245 méthodes. La conformité aux 117 règles est de 85%. Le projet comprend également 5 300 lignes de XML, qui inclues notamment les fichiers de gestion de projet.

DynaMo est divisé en 43 sous-projets. Les 4 plus gros sous-projets en termes de lignes de code sont le *core*, la bibliothèque standard de médiateurs, l'API et l'interface graphique générique. 25 sous-projets sont des mandataires de dispositifs ou d'applications. Comme le protocole D-Bus est présent dans la plateforme, il est très simple d'ajouter plus d'une centaine de logiciels à contrôler.

Le core a été particulièrement surveillé. Il contient 10 500 lignes dont 5 200 de codes, 73% des API sont documentées. Il contient 100 classes et 472 méthodes. La conformité aux 117 règles est de 91%.

5.4.2 Une implémentation fonctionnelle

DynaMo consiste en un gestionnaire autonomique et une plateforme d'intégration. La plateforme d'intégration inclut Cilia et Rose. Il fonctionne sur OSGi. Plusieurs protocoles sont actuellement pris en compte par DynaMo, ainsi que plusieurs dispositifs et applications.

Le chapitre suivant présente quelques aspects de DynaMo à l'exécution : discussion sur ses performances et commentaires sur des exemples d'interfaces générées par DynaMo.

25. Site officiel : `http://www.sonarsource.org/`.

Chapitre 6

DynaMo à l'exécution

Le chapitre précédent a présenté la plateforme logicielle DynaMo, une mise en œuvre logicielle de notre proposition conceptuelle présentée au chapitre 4. Dans ce chapitre, nous complétons la description de la plateforme DynaMo en traitant les aspects liés à l'exécution, selon deux facettes complémentaires : nous traitons d'abord des performances temporelles de la plateforme à la section 6.1, puis nous présentons plusieurs exemples d'interfaces multimodales produites par DynaMo à la section 6.2.

6.1 Évaluation des performances techniques

Du point de vue de l'utilisateur, le principe du retour d'information immédiat a un impact important sur l'utilisabilité : aussi la rapidité de réaction de l'interface multimodale est un enjeu majeur. Au sein de la plateforme DynaMo, nous distinguons deux temps de réaction qui ont un impact sur la rapidité du retour d'information (indépendamment du temps de réaction de l'application) :

- **temps de propagation d'une donnée** entre un dispositif et une application ;
- **temps d'adaptation d'une interface** entre le moment où un changement est détecté dans l'environnement et le moment où une donnée peut transiter entre un dispositif et une application.

Les deux sections suivantes sont consacrées à ces deux durées.

6.1.1 Temps de propagation des données

Un temps de propagation des données trop lent réduit l'utilisabilité de l'application : en effet cette dernière semble « en retard » ou décalée. [Dabrowski et Munson, 2001] annoncent entre 150 et 195 millisecondes en moyenne la durée minimum pour qu'un utilisateur perçoive un délai entre son action et le retour visuel de cette action. Ce temps varie selon l'utilisateur et la tâche qu'il doit réaliser.

Étant donnés un service, un ou des dispositifs, une application et un traitement effectué sur les données au sein d'une interface multimodale, le délai induit par DynaMo est égal au délai induit par chaque médiateur Cilia [1]. Ce délai a été évalué à moins de 2 millisecondes pour la traversée de 10 médiateurs de bout en bout [2] [Garcia, 2012, section « mesures de l'impact de l'exécution »],

1. En effet, la partie médiation de données est réalisée par Cilia, voir la section 5.2.2 page 104.
2. C'est-à-dire $0,2$ millisecondes par médiateur en moyenne.

ce qui est négligeable au regard d'une part du délai minimum de perception d'un utilisateur, et d'autre part à la somme de la durée passée entre le dispositif et la chaine de médiation, de la durée de traitement « métier » effectué sur les données dans les médiateurs, de la durée passée entre la chaine de médiation et l'application, et de la durée de traitement par l'application.

Nous soulignons que pour tous les exemples de chaines de médiation que nous avons traités au sein de DynaMo, aucune donnée n'a eu à traverser plus de dix médiateurs entre son entrée et sa sortie de chaine de médiation.

L'implémentation de DynaMo utilise souvent des médiateurs « identité » qui n'ont aucun rôle métier : la donnée qui entre dans un tel médiateur ressort immédiatement sans subir de transformation. Ces médiateurs sont retirés juste avant l'instanciation d'un modèle de chaine de médiation, ils n'ajoutent donc pas de délai lors de la transition d'une donnée.

Nous venons de montrer qu'il est improbable que DynaMo – via Cilia – introduise de délai qui impacterait négativement la fluidité de l'interaction. Aussi nous étudions dans la section suivante l'autre délai possible induit par DynaMo, entre la découverte d'un changement au sein de l'environnement et la modification d'une interface multimodale.

6.1.2 Temps d'adaptation

Le temps d'adaptation correspond à la durée entre la détection d'un changement dans l'environnement et le moment où une donnée peut transiter dans cette interface. Ce délai est moins critique que le précédent dans le sens où dans la plupart des cas, l'utilisateur passe plus de temps à interagir avec une application qu'à attendre la reconfiguration d'une interface multimodale.

Ce temps correspond à la somme de :

- la durée entre le changement et sa détection par DynaMo ;
- la durée du travail du gestionnaire autonomique ;
- la durée de l'instanciation de la chaine de médiation par Cilia.

La durée entre le changement et sa détection est très dépendante du protocole de découverte. Par exemple, des requêtes sont sans cesse renouvelées à la pile réseau Bluetooth pour découvrir les dispositifs, la découverte de dispositifs Bluetooth est de l'ordre de la seconde [Peterson *et al.*, 2006]. D-Bus, un protocole de communication interprocessus local est plus de l'ordre de la milliseconde ou de la dizaine de millisecondes.

L'instanciation d'une chaine de médiation par Cilia est très lente : l'instanciation d'un médiateur est de 130 millisecondes. Un chaine de médiation générée par DynaMo contient typiquement une vingtaine de médiateurs.

Tandis que DynaMo a une emprise indirecte sur la durée de l'instanciation qui dépend de Cilia, DynaMo a une emprise directe sur la durée du travail du gestionnaire autonomique. L'explosion combinatoire des possibilités est inhérente à la multimodalité en environnement pervasif de part son opportunisme : il existe des milliers de dispositifs, et ceux-ci peuvent être librement combinés. Le gestionnaire autonomique est donc à priori sujet à cette explosion combinatoire.

Pour faire face à cette explosion combinatoire, l'algorithme a été écrit de manière à ce que sa complexité soit dépendante du nombre de dispositifs d'interaction présents dans l'environnement plutôt que du nombre de modèles disponibles. Plus précisément, la durée du travail du gestionnaire autonomique est dépendante de deux facteurs : le nombre de dispositifs présents dans l'environnement, et la structure des dépendances entre les modèles. Le pire cas apparait lorsque le graphe des modèles des dispositifs présents et de leur dépendance constitue une clique. Si cette condition est vraie, alors la complexité de l'algorithme est de $O(2^n)$ où n représente le nombre de dispositifs présents. Cependant, cette condition n'a pas de sens, excepté pour un ensemble composé d'un nombre très réduit de modèles (deux ou trois).

À chaque fois que DynaMo détecte un changement dans l'environnement, il génère une nouvelle interface. Durant la génération d'une nouvelle interface, l'interface courante n'est pas arrêtée. En conséquence, si un dispositif d'interaction disparait, l'interface multimodale continue de fonctionner. Le temps d'indisponibilité est réduit au temps pour Cilia d'instancier la nouvelle chaine de médiation. De même, si un dispositif d'interaction apparait, l'interface précédemment produite par DynaMo continue de fonctionner jusqu'à ce que Cilia doive instancier la nouvelle chaine.

6.1.3 Discussion sur les performances

Tout au long de l'élaboration des propositions conceptuelles et logicielles, la performance a été un critère central. L'hypothèse de milliers – voire de centaines de milliers – de modèles présents dans les dépôts a été posée. L'explosion combinatoire des possibilités est donc inhérente à ce contexte de multimodalité en environnement pervasif. Les techniques mises en œuvre visent à réaliser des prétraitements rapides qui réduisent drastiquement les facteurs de l'explosion combinatoire, et la mise en place de garde-fous pour les rares cas où les facteurs ne seraient pas suffisamment réduits.

Pour des raisons de sécurité, l'utilisateur doit autoriser un type de dispositif avant que celui-ci puisse contrôler une application via DynaMo. Le gestionnaire autonomique connait uniquement les modèles de mandataire préalablement autorisés par l'utilisateur. Cela maintient le nombre de modèles disponibles très bas.

La nécessité de ne pas allonger démesurément la durée de transition d'une donnée dans DynaMo a proscrit le calcul de chemins pour chaque donnée entrante dans l'interface, au profit d'une interface calculée une seule fois jusqu'au changement suivant dans l'environnement.

D'autres aspects de DynaMo lui permettent d'être efficace. Par exemple, la recherche de dispositifs et d'applications se fait par protocole, et non pas par dispositif – contrairement à d'autres plateformes pour les interfaces multimodales comme ICon, ICARE, OpenInterface décrites dans la section 3.2. Cela réduit drastiquement le nombre de processus (*threads*) nécessaires.

DynaMo a été instrumenté pour mesurer le temps de l'adaptation. Une génération en masse de modèles devrait permettre de tracer des courbes et confirmer cette discussion. Cette instrumentation a pour l'instant été utilisée pour des mesures sur un petit nombre de modèles uniquement. Aucune anomalie n'a été détectée. Ces modèles et les interfaces générées par DynaMo font l'objet de la section suivante.

6.2 Exemples développés avec DynaMo

Cette section illustre le fonctionnement de la plateforme DynaMo en présentant un ensemble d'exemples complémentaires qui fonctionnent avec Dynamo. Pour chaque exemple est décrit l'état de l'environnement, les modèles disponibles, et l'interface générée par DynaMo. Ces exemples sont regroupés en trois parties qui correspondent à trois cas de figure qui se présentent à DynaMo :

- DynaMo ne dispose d'aucune information : il connait uniquement les tâches d'une application et les capteurs des dispositifs présents dans l'environnement. Ce cas se produit systématiquement s'il existe uniquement des modèles de mandataire.
- DynaMo dispose d'informations complètes : il dispose d'un modèle décrivant précisément comment lier les tâches et les capteurs des dispositifs présents. Ce cas produit des interfaces

très bien adaptées car entièrement conçues par un concepteur humain. Ce cas se produit systématiquement s'il existe pour tout ensemble de modèles de mandataire un fragment d'interaction qui décrit l'interface. Logiquement, ce cas se produit rarement.
- DynaMo dispose d'informations génériques : il s'appuie sur les applications ou les dispositifs virtuels pour générer une interface. DynaMo est optimisé pour ce cas, qui correspond à un bon compromis entre le nombre de fragments d'interaction à créer par les concepteurs d'interaction et la bonne adéquation des interfaces générées.

Dans les sections suivantes, nous présentons des exemples d'applications multimodales opérationnelles avec DynaMo, qui illustrent ces trois cas. Il convient de noter qu'au sein d'un environnement pervasif dans lequel il existe plusieurs dispositifs, les trois cas peuvent se produire en même temps. Pour la clarté de la présentation, nous avons néanmoins considéré ces trois cas distinctement dans les sections suivantes.

Chacun de ces trois cas passés en revue est à son tour divisé en deux : un exemple avec un seul dispositif qui permet de comprendre facilement comment travaille le gestionnaire autonomique, puis un exemple avec plusieurs dispositifs, puisque DynaMo est une proposition pour la génération d'interfaces exploitant la multiplicité des dispositifs.

Enfin, pour décrire les exemples, deux types de figures sont produits : des fragments d'interaction et des interfaces générées.

Un fragment d'interaction est représenté avec un fond gris. Un fragment d'interaction peut avoir des dépendances sur d'autres fragments d'interaction. Ces dépendances sont alors représentées par des flèches en pointillés. Les fragments d'interaction peuvent contenir des instances de médiateurs qui sont représentées avec un fond blanc. Un port sur la gauche d'une instance est systématiquement un port d'entrée, et un port sur la droite d'une instance un port de sortie. Un fragment d'interaction peut avoir des liaisons, elles sont représentées par des traits noirs qui relient deux ports. Comme expliqué dans la section 4.3.2 page 66, les liaisons d'un fragment peuvent référencer des ports qui existent dans les fragments dont il dépend.

Une interface générée correspond à un modèle de médiation obtenu par le gestionnaire autonomique. Une interface est obtenue à partir des fragments d'interaction présents. Les dispositifs d'interaction sont systématiquement placés sur la droite et l'application placée sur la gauche.

La représentation des instances de composants et des instances de médiateurs est assez libre. Deux ports qui sont sémantiquement proches sont accolés : dans le cas général ils font partie d'un même groupement sémantique tel que défini dans la section 4.5.3.3 page 82.

6.2.1 Présentation des dispositifs et des applications

Pour les exemples présentés dans les sections suivantes, plusieurs applications et plusieurs dispositifs sont considérés. Nous les listons dans cette section introductive. Les deux applications sont :

KSudoku logiciel de sudoku ; le sudoku est un jeu de réflexion qui a pour principe de remplir les cases d'une grille avec des nombres de 0 à 9, en respectant certaines règles. Ce logiciel à deux notions importantes :

case sélectionnée case courante mise en surbrillance dans laquelle on peut marquer un nombre. Pour qu'un chiffre puisse être écrit dans une case, cette case doit nécessairement être sélectionnée.

valeur sélectionnée nombre courant entier parmi l'ensemble [0,9].

Les huit tâches de ce logiciel sont :

clearCell efface le nombre précédemment marqué dans la case sélectionnée.

moveUp, moveDown, moveLeft, moveRight désélectionne la case sélectionnée, et sélectionne respectivement la case du dessus, du dessous, de gauche ou de droite.

setSelectedValue (paramètre : entier $\in [0, 9]$) change le nombre sélectionné par le nombre donné en paramètre.

enterSelectedValue marque le nombre sélectionné dans la case sélectionnée.

enterValue (paramètre : entier $\in [0, 9]$) marque le nombre donné en paramètre dans la case sélectionnée.

VLC un lecteur multimédia ; ses tâches élémentaires sont :

play démarre la lecture si le lecteur est en pause, ou met en pause si la lecture est en cours.

setVolume : permet de régler le volume via un nombre entier passé en paramètre, ce nombre doit être inclus dans l'intervalle [0,100].

stop arrête la lecture du média en cours.

next, previous passe au média suivant ou précédent dans la liste de lecture.

Les quatre dispositifs sont :

Bouton Rotatif bouton fournissant des données entières dans l'intervalle [0,255] ;

Joystick manche posé sur un socle offrant cinq capteurs :

jUp, jDown, jLeft, jRight suivant l'orientation du manche ;

fire bouton.

BDRC acronyme de BD Remote Control, une télécommande Bluetooth orientée vers le contrôle de lecteur de média, et disposant de très nombreux boutons dont :

play, stop, pause, next, previous nombreux boutons de contrôle de lecteur vidéo ;

0, . . . , 9 , nombres de 1 à 9 ;

up, down, left, right quatre boutons qui composent un pavé directionnel ;

triangle, square, circle, cross quatre boutons ayant chacun une forme géométrique représenté dessus.

Wiimote diminutif de Wii Remote, une manette de console de jeu vidéo. Ces 11 capteurs sont :

padUp, padDown, padLeft, padRight quatre boutons qui composent un pavé directionnel ;

a, b, +, -, home, 1, 2 divers boutons ;

x, y, z accéléromètre tridimensionnel. Nombres entiers dans l'intervalle [0,255] ;

roll, pitch, yaw gyroscope tridimensionnel, données calculées par une bibliothèque logicielle à partir des valeurs de l'accéléromètre. Nombres flottants dans l'intervalle [-180,180].

Les exemples des sections suivantes considèrent ces deux applications et ces quatre dispositifs. Ces exemples illustrent des interfaces générées en fonction de l'environnement et des fragments d'interaction disponibles. Il est important de noter que les deux applications sont des applications existantes disponibles en téléchargement et n'ont donc pas été développées de façon à fonctionner avec DynaMo.

6.2.2 Génération d'interfaces : absence d'information

Dans certains cas, DynaMo doit faire face à l'absence totale d'information sur l'interaction : il connait systématiquement les tâches d'une application, et les capteurs et effecteurs d'un ou des dispositifs d'interaction. Ce cas se produit donc lorsque DynaMo ne dispose pas de fragments d'interaction qui le guideraient dans la création de l'interface entre les dispositifs et l'application. Les deux exemples suivants illustrent ce cas.

Les processus mis en œuvre pour ce cas sont décrits dans la section 4.5.3.2 page 81. Ces processus qui définissent des connexions entre des ports sont appliqués dans le cas où aucun modèle d'interaction utile n'est trouvé, mais aussi pour la connexion de tout port libre d'un dispositif.

6.2.2.1 Dispositif unique

Un exemple d'interface générée sans fragment d'interaction est représenté sur la figure 6.1. L'application *A* a deux groupements sémantiques : [x, y, z] et [m, n], le dispositif a également deux groupements sémantiques : [1, 2, 3] et [a, b]. Ces dix ports sont donc reliés ensemble. Les ports *f01* et *i1010* peuvent être reliés en insérant un médiateur de conversion de types et un médiateur d'adaptation d'intervalles. Les ports *temp* et *t* sont reliés car leur type de capteur sont identiques, à l'inverse de *xy* et *gps* qui ne sont donc pas reliés.

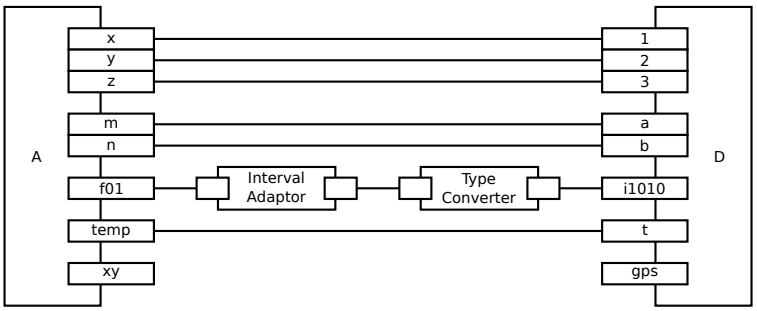


FIGURE 6.1 – Interface générée permettant de contrôler l'application A avec le dispositif D en l'absence de toute information. Les ensembles [x, y, z], [m, n], [1, 2, 3] et [a, b] sont quatre groupements sémantiques. *f01* accepte des flottants inclus dans l'intervalle [0, 1], *i1010* fournit des entiers inclus dans l'intervalle [-10, 10]. Les ports *temp*, *t* et *gps* possèdent chacun un type de capteur, les deux premiers sont de type « thermomètre », le dernier est de type « GPS ». Les ports *xy* et *gps* sont compatibles.

6.2.2.2 Plusieurs dispositifs

Un autre exemple d'interface générée en l'absence de fragment d'interaction est présenté sur la figure 6.2. Cet exemple considère deux dispositifs.

DynaMo est capable de relier deux sources de booléens vers une destination acceptant des nombres : il insère un composant *Event to Number Converter* qui possède un compteur interne. Ce compteur est incrémenté ou décrémenté à chaque arrivée de données sur ses ports + et -. À chaque changement d'état, la nouvelle valeur du compteur est émise sur la sortie.

DynaMo est également capable d'insérer un médiateur de fusion pour lier un port nécessitant une liste de valeur à des ports fournissant chacun une valeur. Puisqu'elle est générée automatiquement, cette fusion n'est pas systématiquement utile. En effet, un composant de fusion peut

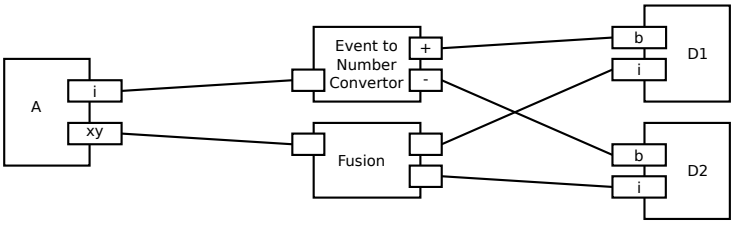


FIGURE 6.2 – Interface générée permettant de contrôler l'application A avec les dispositifs D1 et D2 en l'absence de toute information sur l'interaction. Les deux ports *b* fournissent des booléens, les deux ports *i* des dispositifs fournissent des entiers, le port *i* de l'application accepte des entiers, le port *xy* accepte des couples d'entiers.

être configuré de différente manière (par exemple en spécifiant la manière de synchroniser les données), le médiateur de fusion inséré automatiquement reçoit une configuration générique qui peut ne pas être bien adaptée.

La configuration par défaut fusionne les données dans une fenêtre temporelle de trois secondes : si une donnée est arrivée depuis plus de trois secondes et n'a pas été fusionnée, alors elle est abandonnée. Les ports n'ont pas de file d'attente : si une nouvelle donnée arrive sur un port et qu'une donnée est déjà arrivé, alors cette nouvelle donnée remplace l'ancienne.

L'insertion de fusion de données n'est bien sûr pas réservée à des ports de dispositifs différents, les données de deux ports d'un même dispositif peuvent être fusionnées.

6.2.3 Génération d'interfaces : absence de généricité

À l'opposé du cas où DynaMo n'a pas d'information sur la relation entre des dispositifs et une application, DynaMo peut disposer d'un ou plusieurs fragments qui définissent une interface entre un ou plusieurs dispositifs particuliers et une application particulière.

Un exemple d'interface générée est présenté sur la figure 6.4 : il correspond au cas où le gestionnaire autonomique a été guidé par un fragment d'interaction pertinent qui dépend de KSudoku, Joystick et Bouton Rotatif. Ce fragment d'interaction est présenté sur la figure 6.3.

L'interface générée est très similaire au modèle, seul un médiateur d'adaptation d'intervalle est nécessaire pour que les données du port *angle* comprises dans l'intervalle [0, 255] soient transformées en données comprises dans l'intervalle [1,9]. Comme il n'existe aucun port libre appartenant à un dispositif, le gestionnaire autonomique échoue à lier le port *enterValue* qui reste donc libre.

Disposer de modèles décrivant complètement une interface est bien pris en compte par DynaMo. Ce cas correspondant à la définition complète de l'interaction lors de la conception est géré par la plupart des plateformes logicielles dédiées à l'interaction multimodale (comme celles décrites dans la section 3.2). Dans le cas d'un environnement pervasif, nous avons souligné à la section 4.2.2.2 page 64, que tout n'est pas défini à la conception et qu'une explosion combinatoire des possibilités se produit. Le cas où tout est figé à la conception est donc rare si l'on considère des environnements pervasifs. Le cas suivant est à l'inverse amené à se produire beaucoup plus souvent.

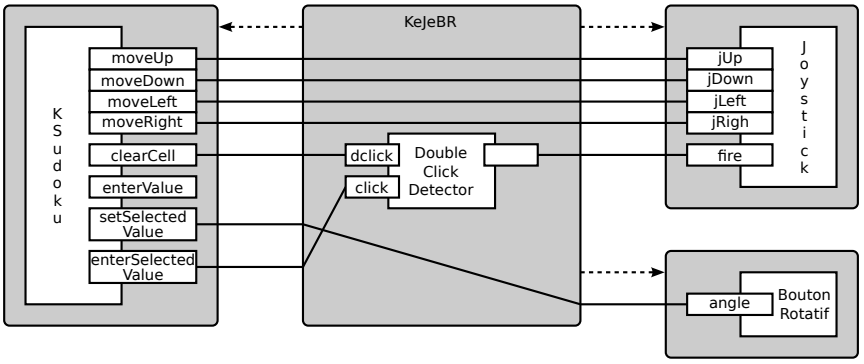


FIGURE 6.3 – Fragment d'interaction nommé KeJeBR qui dépend de KSudoku, Joystick et Bouton Rotatif. Ce fragment contient huit liaisons et une instance du composant Double Click Detector. Ce composant permet de différencier les évènements très proches temporellement : à chaque évènement arrivant sur son port d'entrée, il envoie un évènement sur sa sortie *click*, à moins qu'un autre évènement arrive après un délai très court : il envoie alors un évènement sur sa sortie *dclick*.

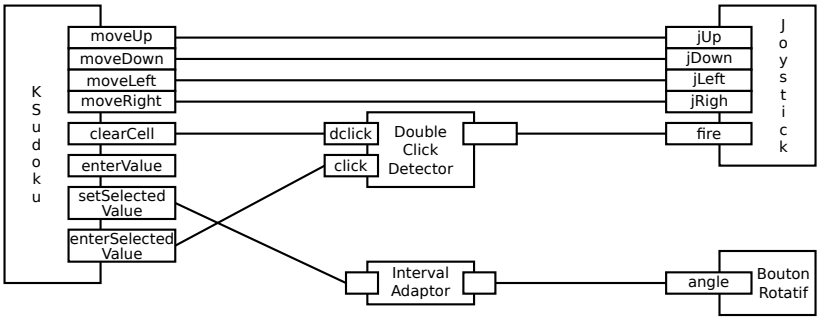


FIGURE 6.4 – Interface générée pour contrôler KSudoku de façon bimanuelle avec un Joystick et un Bouton Rotatif. Pour générer cette interface, le gestionnaire autonomique s'appuie sur le fragment d'interaction représenté sur la figure 6.3.

6.2.4 Génération d'interfaces : généricité

Les cas précédemment présentés n'utilisent pas les concepts d'applications et de dispositifs virtuels. Ces concepts permettent un juste milieu entre l'absence d'information sur l'interaction pour guider la génération de l'interface, et la nécessité de produire une spécification pour chaque état possible des environnements pervasifs : ils introduisent de la généricité dans la description des interfaces.

Cette section présente deux exemples d'interfaces générées en partie grâce à des applications ou dispositifs virtuels. Le premier exemple considère un seul dispositif, une télécommande notée BDRC, tandis que le deuxième exemple deux dispositifs, une Wiimote et une télécommande BDRC.

6.2.4.1 Dispositif unique

L'interface générée est représentée sur la figure 6.7. Le gestionnaire autonomique utilise les fragments d'interaction représentés sur les figures 6.5 et 6.6.

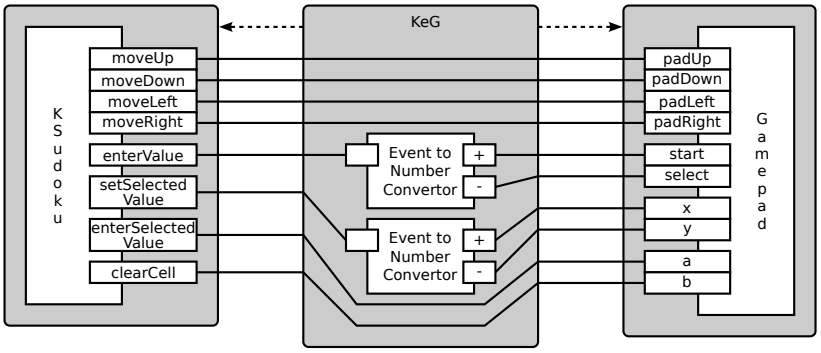


FIGURE 6.5 – Fragment d'interaction entre KSudoku et le dispositif virtuel Gamepad. Nommé KeG, il contient deux instances de composants et douze liaisons.

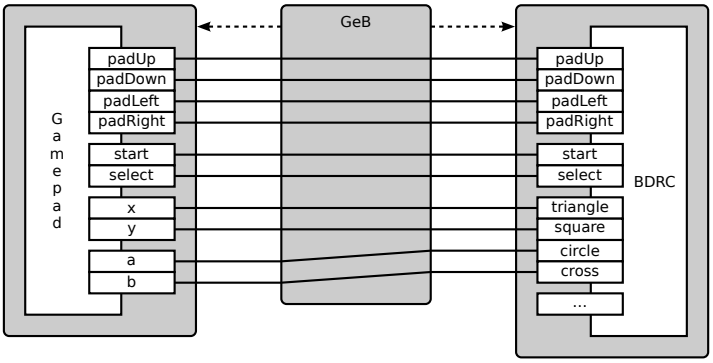


FIGURE 6.6 – Fragment d'interaction entre le dispositif virtuel Gamepad et la télécommande BDRC. Nommé GeB, il contient dix liaisons et aucune instance de composant. Le modèle de la BDRC n'est pas complètement représenté.

Dans cet exemple, le dispositif virtuel est noté Gamepad. Ce dispositif virtuel constitue une représentation intermédiaire entre le modèle de KSudoku et celui de la télécommande notée BDRC. En effet, un fragment d'interaction (figure 6.5) définit la correspondance entre les concepts du dispositif Gamepad et ceux de KSudoku tandis qu'un autre fragment (figure 6.6) définit la correspondance entre les concepts du dispositif Gamepad et ceux de la BDRC. Même s'il n'existe plus de « frontière » une fois l'interface générée, la manière qu'ont les données de transiter au sein de cette interface est contrainte par le dispositif Gamepad.

L'interface générée possède un grand nombre de liaisons qui ne sont pas représentées sur la figure 6.7 à des fins de lisibilité. L'utilisateur peut néanmoins sélectionner une règle pour indiquer de restreindre l'activation de chacune des tâches de l'application par des actions captées par un seul capteur : cette règle implique que DynaMo ne va construire les équivalences (au sens des propriétés CARE chapitre 2) de capteurs pour chaque tâche lors de la transformation de modèle [3].

La mise en œuvre de cet exemple peut sembler plus couteuse que celle de l'exemple sans généricité de la section 6.2.3 où un seul fragment d'interaction a été défini. En effet si l'environnement est stable et connu à l'avance, l'approche par des interfaces génériques est inutile : la solution sans généricité illustrée à la section 6.2.3 est alors optimale. À l'opposé, dans des

3. Voir la section Transformation : exploitation des ports libres des dispositifs et des applications page 81 pour plus de détails.

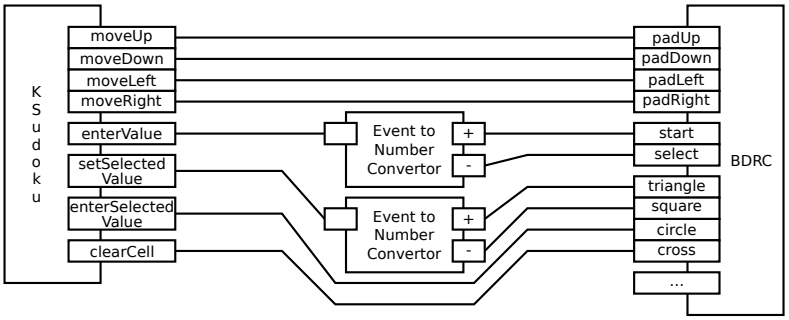


FIGURE 6.7 – Interface générée entre KSudoku et la BDRC en utilisant les fragments d'interaction KeG et GeB qui sont présentés sur les figures 6.5 et 6.6.

environnements pervasifs, la solution avec des interfaces génériques définit une approche adaptée pour gérer la variabilité et le dynamisme inhérents aux environnements pervasifs et pour mettre en œuvre un comportement opportuniste de la plateforme. Ce comportement opportuniste de DynaMo est illustré dans cet exemple : en effet les deux fragments d'interaction KeG (figure 6.5) et GeB (figure 6.6) ont été conçus de façon totalement indépendante, et peuvent avoir été spécifiés par deux concepteurs. Du point de vue de l'utilisateur qui joue à KSudoku, ceci est complètement transparent et l'utilisateur peut penser que l'application KSudoku a été conçue par un seul concepteur et pour être manipulée avec la télécommande BDRC.

Enfin, l'interface générée illustre un aspect de DynaMo qui a été discuté dans la section intitulée Greffe de multimodalité : notre hypothèse page 35 : l'hypothèse dans nos travaux est un accès à une application par son API uniquement, sans accès et modification du code. Si l'API de l'application ne donne pas accès de façon détaillée à l'état de l'application, des problèmes interactionnels au niveau de l'interface multimodale en entrée, générée par DynaMo, peuvent apparaitre. Ceci est le cas dans notre exemple qui souligne les limites de la plateforme DynaMo. En effet KSudoku dispose d'une interface graphique en sortie qui est confondue avec son noyau fonctionnel et l'API ne donne pas accès à l'état de l'interface graphique en sortie. Ainsi, lorsqu'un utilisateur change la valeur sélectionnée, l'interface en entrée générée par DynaMo n'est pas notifiée. Au prochain évènement produit par x ou y de Gamepad ou concrètement par l'appui d'une des deux touches Triangle et Square de la télécommande BDRC, le résultat ne sera pas conforme à ce qu'attend l'utilisateur, puisque le compteur interne au convertisseur dans l'interface en entrée n'a pas été notifié de la modification de la valeur courante.

L'API de KSudoku est un exemple d'API particulièrement pauvre. En général, une API qui permet de modifier un état permet également de lire cet état. Dans ce cas, il existe plusieurs pistes pour résoudre cette absence de synchronisation entre l'état de l'interface multimodale et l'état interne. Il convient de les étudier plus avant pour les intégrer dans la plateforme DynaMo.

Tout en illustrant la généricité au niveau de la spécification des fragments d'interaction, l'exemple suivant considère deux dispositifs.

6.2.4.2 Plusieurs dispositifs

L'interface générée est représentée sur la figure 6.11. Le lecteur multimédia VLC est contrôlable par trois modalités partiellement équivalentes (au sens des propriétés CARE) pour les cinq tâches élémentaires de VLC. Ces trois modalités reposent sur deux dispositifs en entrée distincts : la télécommande BDRC et la Wiimote. Pour générer cette interface, le gestionnaire autonomique

exploite quatre fragments d'interaction représentés sur les figures 6.6, 6.8, 6.9 et 6.10.

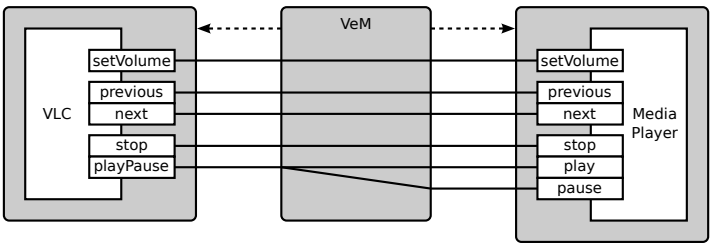


FIGURE 6.8 – Fragment d'interaction entre l'application VLC et l'application virtuelle Media Player.

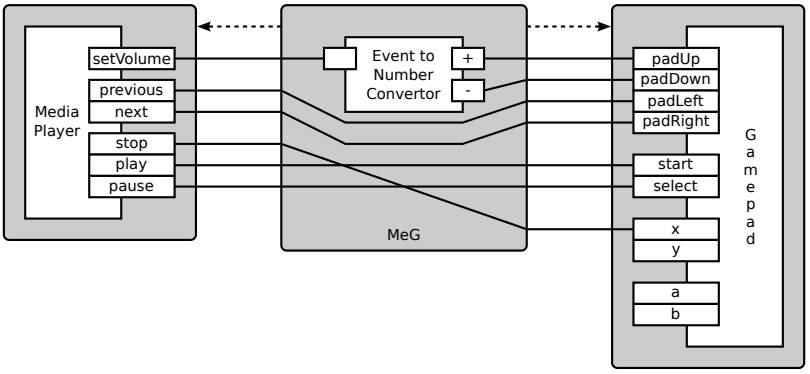


FIGURE 6.9 – Fragment d'interaction entre l'application virtuelle Media Player et le dispositif virtuel Gamepad.

Pour les deux modalités générées reposant sur la télécommande BDRC, la chaine de dépendances entre VLC et la BDRC inclut une application virtuelle (Media Player) et un dispositif virtuel (Gamepad). Ainsi, le fragment d'interaction qui a été utilisé dans l'exemple précédent (section 6.2.4.1) pour contrôler KSudoku avec la BDRC est réutilisé dans cet exemple pour contrôler VLC. Les liaisons des trois ports *square*, *circle* et *cross* de la figure 6.11 sont issues de la recherche de ports libres. Comme pour l'exemple précédent de la section 6.2.4.1, les ports de la BDRC qui ne sont représentés à des fins de lisibilité possèdent aussi des liaisons avec VLC.

Pour la modalité reposant sur la Wiimote, la chaine de dépendance entre VLC et la Wiimote inclut une application virtuelle (Media Player), résultant de l'exploitation des deux fragments d'interaction des figures 6.8 et 6.10. Il est intéressant de noter que l'effet de l'instance Alternator (figure 6.10) est annulé par VLC (figure 6.8).

6.3 Conclusion

Dans ce chapitre nous avons complété la description de notre plateforme logicielle DynaMo du chapitre 5 en étudiant les aspects liés à l'exécution, d'abord en considérant les performances de la plateforme puis en présentant plusieurs exemples d'applications multimodales en entrée.

La plateforme logicielle DynaMo a été développée pour tester et valider les solutions qui sont avancées dans notre proposition conceptuelle. Considérer de nombreux exemples d'interfaces

multimodales a fait partie de notre démarche de travail alliant une approche du haut vers le bas (des concepts vers des applications multimodales concrètes) avec une approche opposée du bas vers le haut en partant d'exemples d'applications multimodales. Les exemples présentés dans ce chapitre sont complétés par d'autres exemples qui ont été décrits dans les articles publiés. Dans ce chapitre nous avons illustré la couverture fonctionnelle de la plateforme DynaMo en considérant trois cas :

1. DynaMo ne dispose d'aucune information pour guider la génération de l'interface multimodale ;
2. Dynamo a à sa disposition une description complète de l'interface multimodale ;
3. Dynamo dispose d'informations génériques et parcellaires sur l'interaction qui lui permet de générer une interface multimodale de façon opportuniste.

L'existence de la plateforme DynaMo avec ses nombreux exemples, mettant en jeu des applications existantes (et non développées pour fonctionner avec DynaMo) et plusieurs dispositifs d'interaction, constitue une forme de validation de notre proposition conceptuelle. Cette validation doit être complétée par d'autres formes d'expérimentations qui impliquent des concepteurs d'interfaces et mais aussi des utilisateurs finaux. Ceci constitue des perspectives à nos travaux que nous décrivons dans le chapitre suivant de conclusion générale.

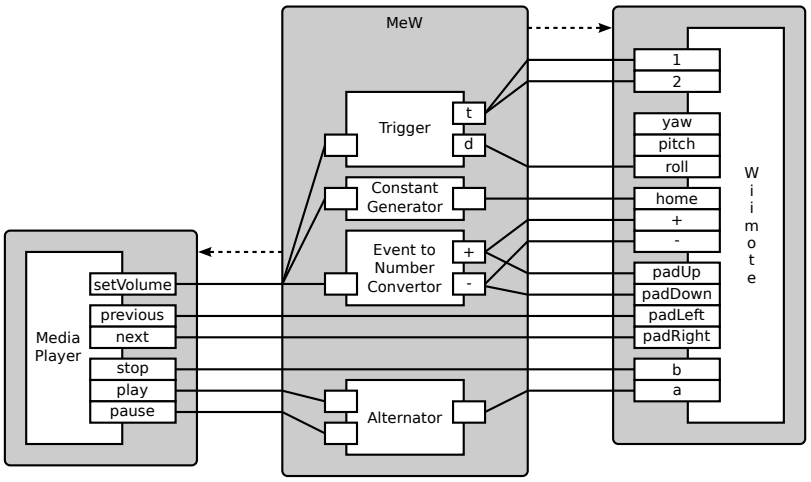


FIGURE 6.10 – Fragment d'interaction entre l'application virtuelle Media Player et la Wiimote.
Le médiateur Trigger (déclencheur en français) envoie sur son port de sortie la dernière donnée qui est arrivée sur son port d'entrée *d* à chaque fois qu'il reçoit une donnée sur son port *t*. Le médiateur Trigger permet donc de lier une source qui envoie des données en continue à un destinataire qui n'est pas prévu pour ce type de transmission et attend des données discrètes. Ainsi, à chaque appui sur le bouton 1 ou le bouton 2, le port *setVolume* reçoit la valeur de l'angle que fait la Wiimote avec le sol.
Le médiateur Constant Generator se configure avec une valeur et un intervalle. À chaque réception d'une donnée sur son port d'entrée, il émet la valeur sur son port de sortie. Dans ce fragment d'interaction, cette instance de médiateur est configurée avec la valeur 0 et l'intervalle [0,1]. Ainsi, le port d'entrée de l'instance de ce médiateur constitue une tâche « couper le son », puisque la valeur envoyée est égale à la borne inférieure de l'intervalle.
Le médiateur Event to Number Convertor se configure avec un intervalle. L'instance de ce fragment est configurée avec un intervalle de 10 valeurs. Ainsi, l'utilisateur n'a qu'une dizaine d'appuis sur les boutons pour passer du volume sonore minimum au maximum.
Puisque le niveau du volume peut être modifié via deux modalités, bien qu'il n'y ai qu'un seul dispositif, l'interface est potentiellement multimodale.
Enfin le médiateur Alternator réémet toutes les données reçues sur son port d'entrée en alternant entre ses deux ports de sortie. Comme l'instance est reliée aux tâches *play* et *pause*, ce médiateur permet de créer un bouton « play-pause ».

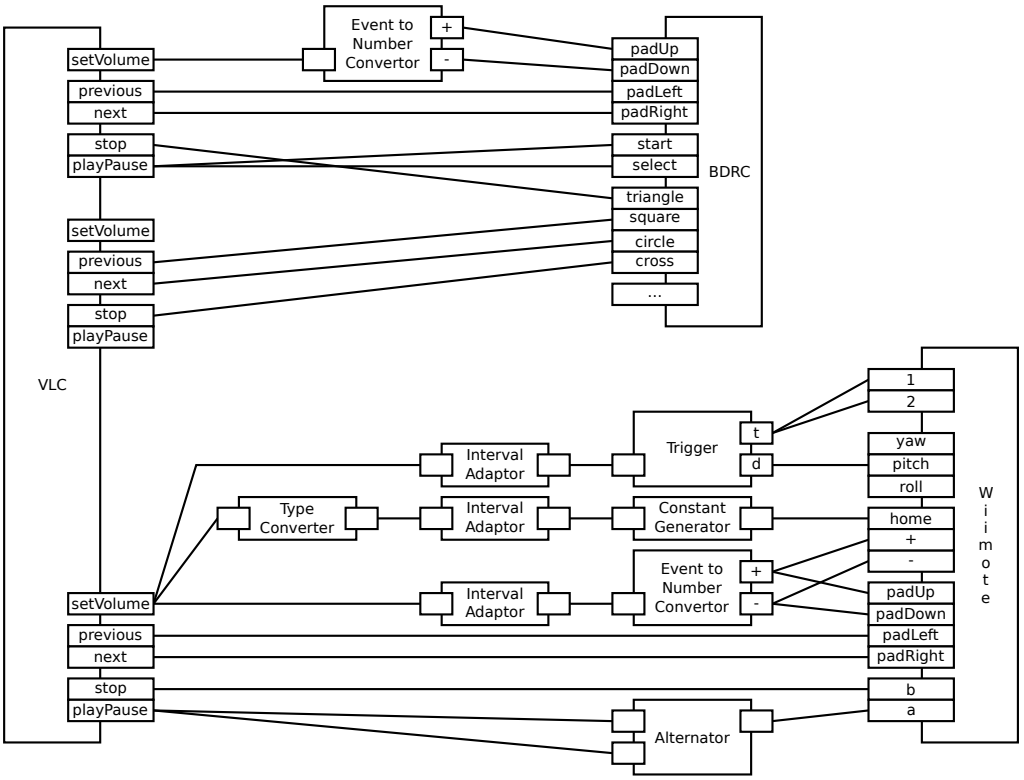


FIGURE 6.11 – Interface générée qui permet de contrôler VLC avec la BDRC et la Wiimote, en utilisant les fragments d'interaction GeB, VeM, MeG et MeW, ce fragments sont présentés sur les figures 6.6, 6.8, 6.9 et 6.10. Les ports de VLC ont été dupliqués pour garder la figure lisible.

Chapitre 7

Conclusion

Partant du constat de la multiplicité des possibilités d'interaction à notre disposition (comme les dispositifs d'interaction variés dans une maison incluant des télécommandes, des manettes de jeu, des téléphones, des tablettes, des objets augmentés) et de la multiplicité des services informatiques destinés à notre quotidien, nos travaux répondent à un problème important aujourd'hui de gestion dynamique et non figée à la conception de l'interaction multimodale dans des environnements pervasifs, qui sont par définition hétérogènes et dynamiques.

Nos travaux constituent des recherches pionnières car pluridisciplinaires en ingénierie de l'interaction multimodale dans des environnements pervasifs. En effet comme nous l'avons souligné dans le chapitre 2, il existe des travaux dans le domaine de l'interaction homme-machine (IHM) qui visent à gérer, de façon partielle et souvent ad hoc, le dynamisme de l'interaction multimodale tandis que des travaux en génie logiciel (GL) proposent des intergiciels pour gérer de façon explicite et efficace l'hétérogénéité et le dynamisme des environnements pervasifs. Néanmoins ces intergiciels ne sont pas dédiés à l'interaction multimodale. Les différentes contributions apportées par cette thèse se situent à la croisée de ces deux domaines IHM et GL et sont récapitulées dans la section 7.0.1.

Notre solution pour l'interaction multimodale dans des environnements pervasifs a été illustrée par des exemples simples d'interaction multimodale qui néanmoins soulignent la large couverture fonctionnelle de notre solution. Il convient maintenant de développer des cas plus complexes et réalistes afin de les tester avec des utilisateurs. Nos réflexions concernant ces cas plus réalistes ainsi que d'autres pistes de recherche pour enrichir et étendre nos contributions sont présentées dans la section 7.0.2.

7.0.1 Contributions de la thèse

Nous avons d'abord souligné les apports centraux de l'interaction multimodale aux environnements pervasifs (chapitre 2). La figure 7.1 résume ces apports et motivent donc nos travaux de recherche.

Dans ce contexte, nos travaux ont été dédiés à l'ingénierie de l'interaction multimodale dans des environnements pervasifs : avant de présenter nos contributions, nous avons montré que les outils existants (chapitre 3) que ce soit ceux en IHM ou ceux en GL (intergiciels) ne répondent pas aux besoins. Cet état de l'art souligne le caractère pluridisciplinaire de nos travaux, comme le schématise la figure 7.2.

Partant du constat sur les outils existants, notre première contribution réside dans la définition conceptuelle d'une plateforme pour la gestion de l'interaction multimodale dans des

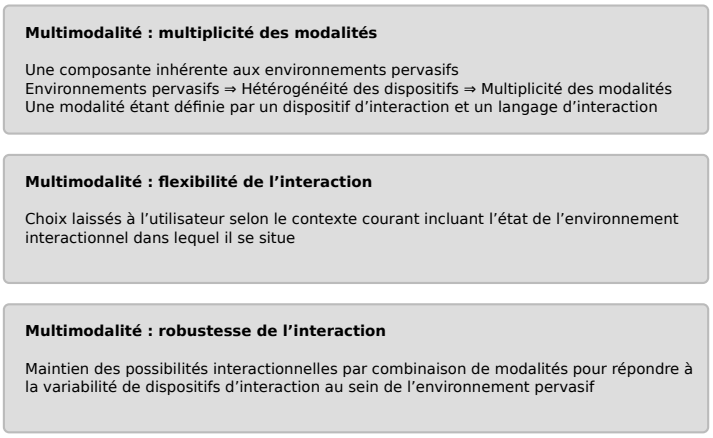


FIGURE 7.1 – Multimodalité et environnements pervasifs : multiplicité, flexibilité et robustesse.

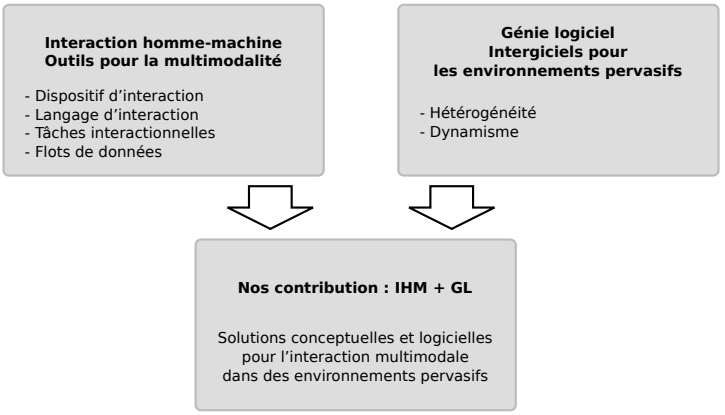


FIGURE 7.2 – Pluridisciplinarité des travaux : outils et plateformes pour la multimodalité et pour les environnements pervasifs.

environnements pervasifs. Notre proposition conceptuelle couvre la spécification et l'exécution d'une interface multimodale.

- La spécification est réalisée par un concepteur d'interfaces qui modélise une interface multimodale. Ces spécifications peuvent être complètes ou partielles.
- L'exécution est réalisée par une plateforme constituée de deux parties distinctes : un gestionnaire autonomique et une plateforme d'intégration. Le gestionnaire autonomique est en charge d'utiliser toutes les informations à sa disposition pour créer un modèle d'interface multimodale adaptée à l'environnement. La plateforme d'intégration est en charge d'exécuter ce modèle et d'assurer la communication entre les applications et les dispositifs disponibles à un instant donné dans l'environnement pervasif.

Notre solution conceptuelle décrite dans le chapitre 4 repose donc sur trois piliers :

- un langage de spécification pour l'interaction multimodale ;
- une plateforme d'intégration ;
- un gestionnaire autonomique.

Ce dernier est en charge de faire le lien entre les applications et les dispositifs d'interaction

qui sont tous deux hétérogènes et dynamiques.

Dans un second temps, nous avons complété nos propositions conceptuelles par leur mise en œuvre logicielle. Ainsi, une deuxième contribution de nos travaux prend la forme d'une plateforme logicielle complète nommée DynaMo : l'implémentation des propositions conceptuelles par un ensemble intégré d'outils logiciels décrit au chapitre 5. En particulier nous avons utilisé deux outils existants pour définir notre plateforme d'intégration : iPOJO et Cilia reposant sur OSGi. iPOJO est le modèle à composant dans lequel sont programmés les mandataires (services applicatifs et dispositifs d'interaction). Cilia constitue la majeur partie de la plateforme de médiation. Cilia reçoit un modèle de chaine de médiation du gestionnaire autonomique et crée une chaine conforme au modèle. Cette chaine constitue l'interface multimodale entre les mandataires.

DynaMo est complètement opérationnel et stable. En particulier, le socle logiciel qui constitue la plateforme d'intégration de DynaMo est robuste. Le chapitre 6 souligne les aspects de performance et fournit un ensemble d'exemples afin d'illustrer le caractère opérationnel de notre plateforme mais aussi souligner sa couverture fonctionnelle. En particulier, DynaMo est capable de créer une interaction face à plusieurs cas :

- en l'absence d'information sur l'interaction ;
- avec des connaissances génériques sur l'interaction ;
- avec une connaissance complète sur l'interaction.

La figure 7.3 résume les compromis entre ces trois cas.

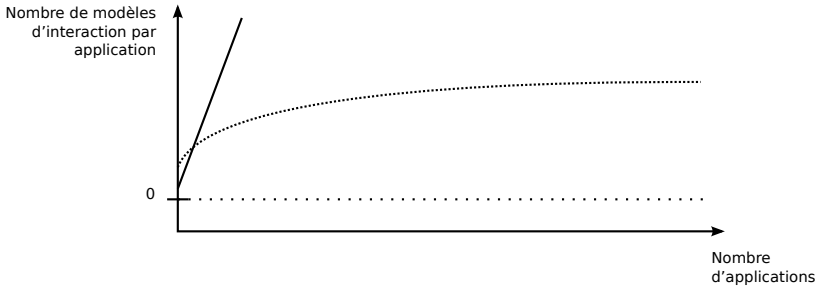


FIGURE 7.3 – Graphe représentant le nombre de modèles d'interaction nécessaires pour construire une interaction selon le nombre d'applications connues : trois cas. Le premier cas – représenté par la courbe en pointillés espacés – ne nécessite aucun travail du concepteur d'interaction, mais l'interface générée peut être peu adaptée à l'utilisateur. Le deuxième cas – représenté par la courbe en pointillés serrés – nécessite du concepteur d'interaction une charge raisonnable de travail et génère en général des interfaces de bonne qualité. Le troisième cas – représenté par la courbe pleine – nécessite du concepteur une quantité de travail qui dépend linéairement du nombre d'applications connues et génère des interfaces très riches et adaptées. Le deuxième cas constitue un compromis mis en avant par DynaMo pour la génération d'interfaces de bonne qualité tout en limitant la quantité de travail à fournir par le concepteur d'interaction.

Selon nos contributions conceptuelles et logicielles, le cycle de développement d'une interface multimodale peut se découper en deux étapes : la description de dispositifs et d'applications pour que la plateforme puisse les prendre en compte, suivie de la description d'une interface multimodale. Après ces étapes, une interface multimodale peut être exécutée. La première étape est réalisée par un développeur, la seconde par un concepteur d'interaction tandis que l'exécution est réalisée par une plateforme logicielle. Notre solution conceptuelle et par conséquent logicielle souligne donc la distinction entre la gestion de l'hétérogénéité et de le dynamisme au niveau de l'intergiciel et la gestion de l'interaction multimodale. Cette distinction explicite dans nos contri-

butions permet d'identifier deux rôles dans l'utilisation d'une plateforme dédiée à l'interaction multimodale dans des environnements pervasifs comme DynaMo :

- Celui du développeur qui définit des mandataires de services applicatifs ou de dispositifs qui sont ensuite gérés dans la plateforme ;
- Celui du concepteur d'interaction multimodale qui définit à haut niveau d'abstraction, indépendamment des aspects implémentationnels, l'interaction multimodale qui exploite des dispositifs pour interagir en entrée avec des services applicatifs.

Nos deux types de contributions, conceptuelles et logicielles, définissent une approche globale pour permettre des interactions multimodales en entrée dans les environnements où les dispositifs d'interaction et les applications sont dynamiques et hétérogènes.

7.0.2 Perspectives de recherche

Comme nous l'avons expliqué, nos travaux constituent des recherches pionnières car pluridisciplinaires en ingénierie de l'interaction multimodale pour des environnements pervasifs. Dans le temps imparti d'une thèse et de par son caractère pluridisciplinaire, nous avons proposé un premier socle conceptuel et logiciel pour la gestion dynamique de l'interaction multimodale : Ce socle constitue une base solide pour de nombreuses pistes de recherche que nous n'avons pas eu le temps d'approfondir. Nous développons trois de ces pistes de recherche qui nous semblent particulièrement prometteuses. Tandis que la première piste de recherche constitue des travaux à mener à court terme car elle vise à exploiter directement nos contributions, les deux autres pistes de recherche constituent des travaux à plus long terme visant à enrichir le socle conceptuel et logiciel que nous avons défini.

7.0.2.1 Peupler la plateforme en vue d'évaluations expérimentales

Un travail à court terme vise à ajouter plusieurs dispositifs d'interaction et plusieurs services au sein de notre plateforme afin de développer des exemples plus conséquents. Ces exemples seront testés par des utilisateurs. L'objectif est de montrer que la plateforme peut gérer des interactions multimodales riches. Pour cela, un premier ensemble d'applications pourrait être basé sur les applications multimodales développées avec les outils existants en interaction multimodale comme les applications de la plateforme OpenInterface [Serrano *et al.*, 2008b] incluant par exemple le paradigme du « mets ça là » pour interagir avec une carte sur un grand écran avec des gestes 3D et la parole. Nous envisageons aussi de tester avec des utilisateurs la multiplicité des modalités possibles et leurs apparitions et disparitions. Néanmoins tester une telle plateforme en vraie grandeur impliquerait une étude longitudinale car son intérêt apparaitrait dans l'appropriation de la plateforme par les utilisateurs au cours du temps ainsi que dans la gestion des cas d'exception comme une panne d'un dispositif d'interaction. Ce travail constitue une perspective à plus long terme et nécessite d'abord d'étendre notre plateforme selon les axes de travail ci-dessous.

7.0.2.2 Enrichir le gestionnaire autonomique

Nous identifions plusieurs pistes de recherche pour enrichir le fonctionnement de notre gestionnaire autonomique.

Tout d'abord le gestionnaire autonomique pourrait exploiter plus d'informations contextuelles [Coutaz *et al.*, 2005]. Par exemple le gestionnaire pourrait apprendre de l'interaction effectuée par l'utilisateur : ses préférences en termes de dispositifs d'interaction et de couplage

de dispositifs à des applications [1]. Cette approche peut être directement développée au sein de notre plateforme puisqu'elle gère l'interaction à un niveau de granularité fine. De plus des informations contextuelles sur la situation d'interaction pourraient aussi guider le gestionnaire autonomique : par exemple choisir les dispositifs d'interaction les plus proches de l'utilisateur (« proxemic interactions » [Greenberg *et al.*, 2011]).

De plus, les connaissances exploitées par le gestionnaire autonomique sous la forme de classes de dispositifs et d'applications (chapitre 4) pourraient être étendues en considérant des ontologies (les classes comme nous les avons introduites constituant des ontologies très basiques). Comme décrit dans [Vanderhulst *et al.*, 2008], de nombreuses ontologies existent pour les environnements pervasifs. Par exemple le projet européen SOFIA (Smart Objects For Intelligent Applications) a travaillé sur des mécanismes basés sur des ontologies pour les environnements pervasifs [Niezen *et al.*, 2010] : ainsi l'ontologie notée « semantic media » dans [Niezen *et al.*, 2012] est très proche de notre classe « media player ». En étendant DynaMo pour manipuler des ontologies, le gestionnaire autonomique définirait la correspondance entre les ontologies des applications et des dispositifs en exploitant des mécanismes d'alignement d'ontologies (comme les types de correspondance : équivalence, inclusion, spécification/généralisation).

7.0.2.3 Rendre observable et contrôlable les décisions du gestionnaire autonomique

Notre solution basée sur un gestionnaire autonomique est prometteuse car elle permet une adaptation automatique à l'environnement pervasif. Sa contrepartie est bien évidemment que l'utilisateur doit connaitre et comprendre les décisions prises par le gestionnaire afin de pouvoir interagir dans l'environnement. Comme souligné dans [Coutaz, 2007], l'utilisateur doit au moins savoir quelles sont les possibilités interactionnelles à un instant donné. De plus l'utilisateur peut vouloir contrôler et diriger les décisions prises par le gestionnaire autonomique : nous sommes dans un contexte d'initiative mixte [Horvitz, 1999] entre l'utilisateur et le gestionnaire autonomique avec différentes modèles de partage d'autorité homme-système qu'il convient d'approfondir.

Ces aspects d'observabilité et de contrôle du gestionnaire autonomique par l'utilisateur impliquent la mise en place d'une interface de contrôle (aussi notée méta-IHM [Coutaz, 2007]) comme le « menu pervasif » de la plateforme ReWire [Vanderhulst *et al.*, 2008]. L'interface de contrôle de DynaMo est pour l'instant très simple et ne répond pas à tous les besoins d'observabilité qu'impliquent la plateforme : en effet l'interface affiche la liste des services applicatifs et des dispositifs disponibles à un instant donné ainsi que les capteurs des dispositifs liés aux ports du service applicatif courant. Cette information est partielle : ainsi les modalités d'interaction ne sont pas décrites. Par exemple le lien entre les accéléromètres de la Wiimote et un port du service applicatif courant est présenté graphiquement mais rien n'est dit sur le geste 3D à effectuer avec la Wiimote pour activer ce port du service applicatif. Pour enrichir cette interface de contrôle, nous avons ébauché une piste de recherche dans la conclusion du chapitre 4 : actuellement le gestionnaire autonomique permet la transformation de concepts d'interfaces multimodales spécifiés selon notre langage d'interaction vers les concepts propres à l'exécution de l'interaction fournis par la plateforme d'intégration. La transformation inverse – des concepts de l'exécution vers les concepts d'interaction – doit être étudiée afin de présenter à l'utilisateur dans des termes d'interaction le modèle de l'exécution pour un contrôle fin des techniques d'interaction et une expression facilitée des préférences interactionnelles.

La solution autonomique que nous avons adoptée dans nos contributions est prometteuse car l'adaptation à l'environnement pervasif est faite automatiquement, l'utilisateur pouvant ainsi se concentrer uniquement sur son intention à réaliser une tâche. Cette solution autonomique

1. Cette fonctionnalité commence à prendre forme : un utilisateur peut déjà classer pour une application les manières de l'utiliser selon ses préférences, voir la section 4.5.2.6 page 79.

séduisante a néanmoins sa contrepartie : mettre l'utilisateur dans la boucle. Ceci constitue un vaste programme de recherche captivant afin de fournir à l'utilisateur les moyens de comprendre et de contrôler le gestionnaire autonomique pour la mise en œuvre dynamique et l'utilisation de techniques d'interaction multimodale dans des environnements pervasifs.

Annexe A

Métamodèles

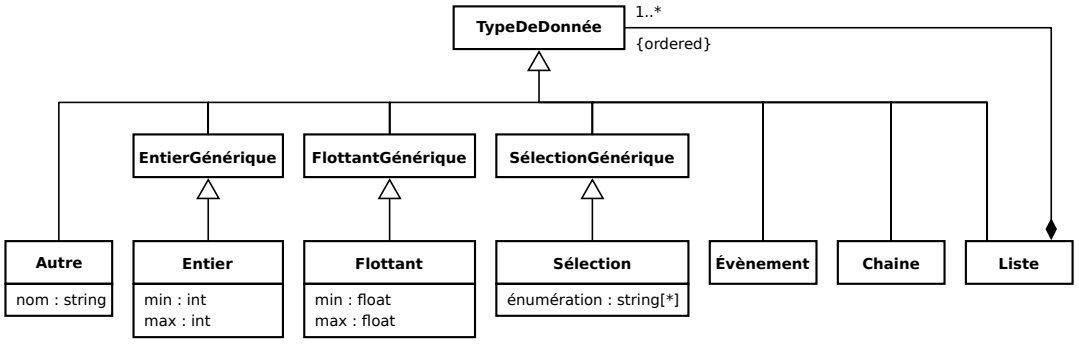


FIGURE A.1 – Métamodèle des types de données.

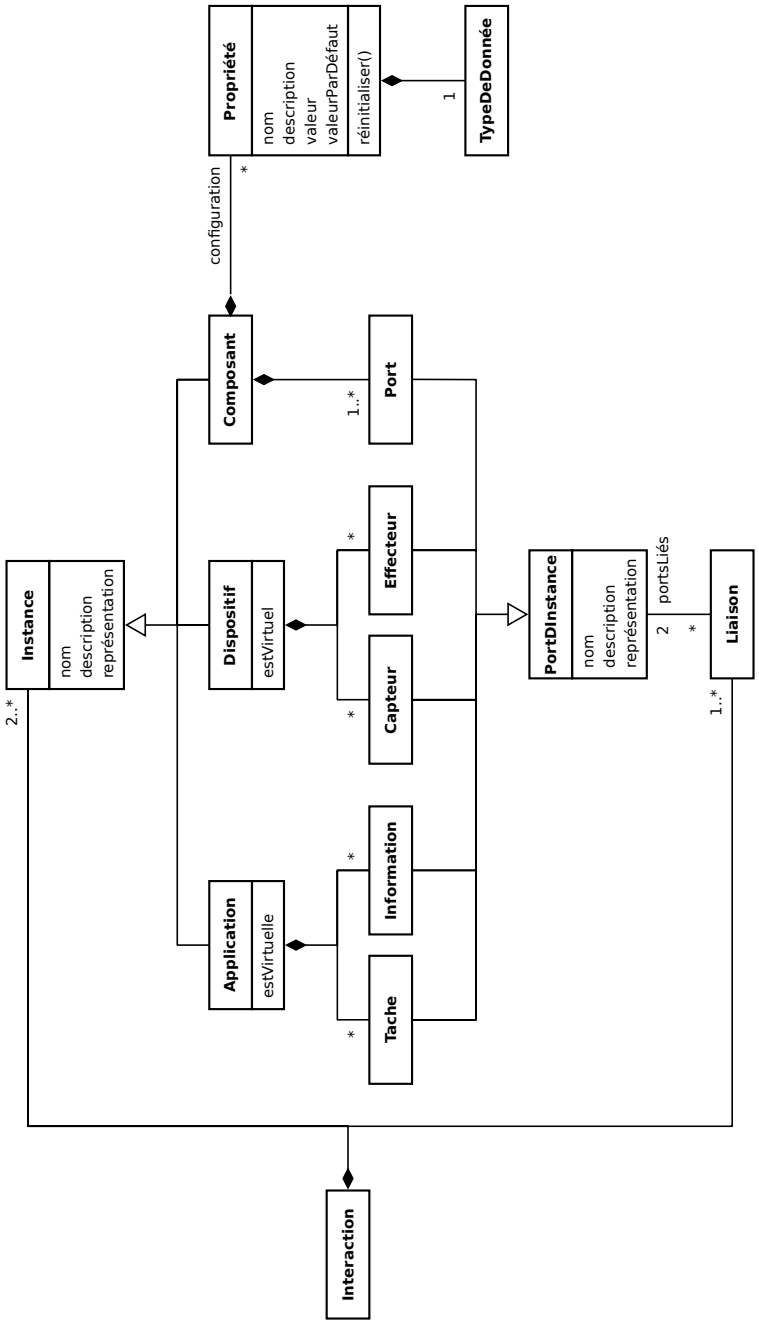


FIGURE A.2 – Métamodèle des interactions pour le concepteur d'interaction.

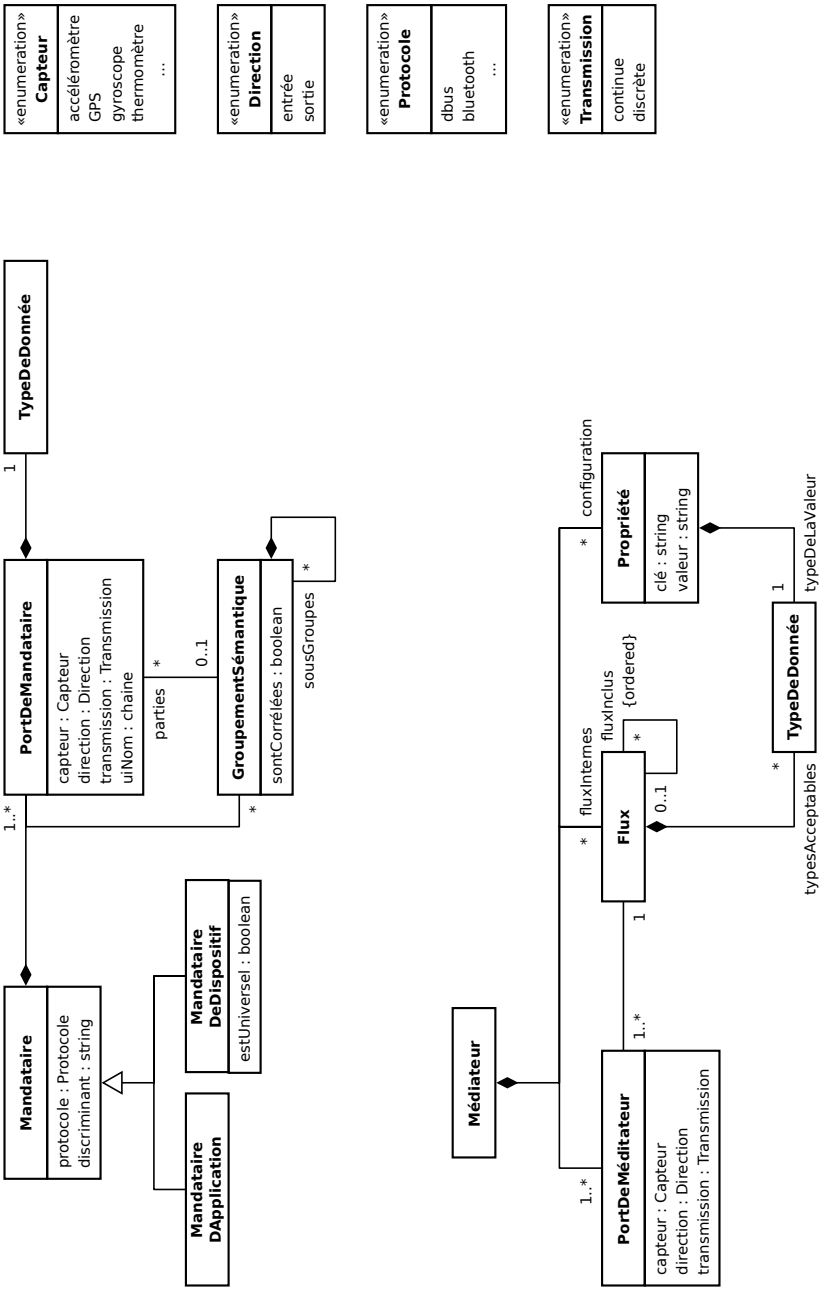


FIGURE A.3 – Métamodèles des mandataires et des médiateurs pour le programmeur. La classe *TypeDeDonnée* est réprésentée sur la figure A.1.

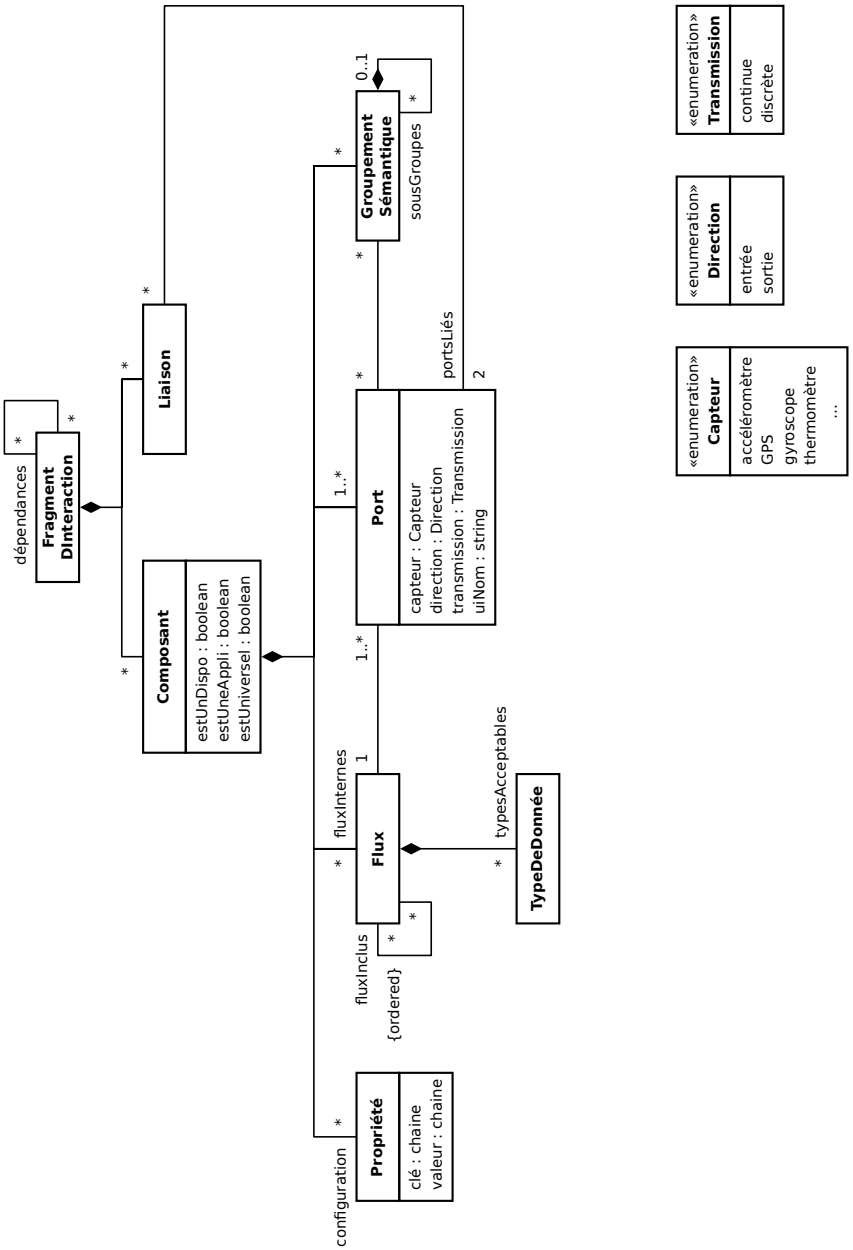


FIGURE A.4 – Métamodèle des fragments d'interaction pour le gestionnaire autonomique. La classe *TypeDeDonnée* est représentée sur la figure A.1.

Bibliographie

[archSigchiBull, 1992] archSigchiBull. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 24(1):32–37, janvier 1992. The contributing authors are (alphabetical order): Len Bass, Ross Faneuf, Reed Little, Niels Mayer, Bob Pellegrino, Scott Reed, Robert Seacord, Sylvia Sheppard and Martha R. Szczur. doi:10.1145/142394.142401.

[Ark et Selker, 1999] Wendy S. Ark et Ted Selker : A Look at Human Interaction with Pervasive Computers. *IBM Systems Journal*, 38(4):504–507, décembre 1999. doi:10.1147/sj.384.0504.

[Avouac *et al.*, 2011a] Pierre-Alain Avouac, Philippe Lalanda et Laurence Nigay : Service-oriented Autonomic Multimodal Interaction in a Pervasive Environment. *In ICMI '11: Proceedings of the 13th International Conference on Multimodal Interfaces*, pages 369–376. ACM, novembre 2011a. doi:10.1145/2070481.2070552.

[Avouac *et al.*, 2012a] Pierre-Alain Avouac, Philippe Lalanda et Laurence Nigay : Adaptable Multimodal Interfaces in Pervasive Environments. *In CCNC '12: Proceedings of the 2012 IEEE Consumer Communications and Networking Conference*, pages 544–548. janvier 2012a. doi:10.1109/CCNC.2012.6181136.

[Avouac *et al.*, 2012b] Pierre-Alain Avouac, Philippe Lalanda et Laurence Nigay : Autonomic Management of Multimodal Interaction: DynaMo in Action. *In EICS '12: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 35–44. ACM, juin 2012b. doi:10.1145/2305484.2305493.

[Avouac *et al.*, 2011b] Pierre-Alain Avouac, Laurence Nigay et Philippe Lalanda : Towards Autonomic Multimodal Interaction. *In MAASC '11: Proceedings of the 1st Workshop on Middleware and Architectures for Autonomic and Sustainable Computing*, pages 25–29. ACM, mai 2011b. doi:10.1145/2034649.2034653.

[Balme *et al.*, 2004] Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz et Gaëlle Calvary : CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. *In* Panos Markopoulos, Berry Eggen, Emile Aarts et James L. Crowley, éditeurs : *Ambient Intelligence*, volume 3295 de *Lecture Notes in Computer Science*, pages 291–302. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30473-9_28.

[Beaudouin-Lafon, 2000] Michel Beaudouin-Lafon : Instrumental Interaction: An Interaction Model for Designing post-WIMP User Interfaces. *In CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 446–453. ACM, 2000. doi:10.1145/332040.332473.

[Berners-Lee *et al.*, 2001] Tim Berners-Lee, James Hendler et Ora Lassila : The Semantic Web. *Scientific American*, 284(5):34–43, mai 2001. doi:10.1038/scientificamerican0501-34.

[Berry, 1988] Richard E. Berry : Common User Access - A Consistent and Usable Human-Computer Interface for the SAA Environments. *IBM Systems Journal*, 27(3):281–300, 1988. doi:10.1147/sj.273.0281.

[Blair *et al.*, 2009] G. Blair, N. Bencomo et R.B. France : Models@run.time. *Computer*, 42(10):22–27, octobre 2009. doi:10.1109/MC.2009.326.

[bluetooth4.0, 2010] *Bluetooth Specification*, juin 2010. Version 4.0. `https://www.bluetooth.org/Technical/Specifications/adopted.htm`.

[bluetoothhid10, 2003] *Bluetooth Human Interface Device (HID) Profile*, mai 2003. Version 1.0 adopted. `https://www.bluetooth.org/Technical/Specifications/adopted.htm`.

[Bolt, 1980] Richard A. Bolt : "Put-That-There": Voice and Gesture at the Graphics Interface. *In SIGGRAPH '80: Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, pages 262–270. ACM, 1980. doi:10.1145/800250.807503.

[Bolt, 1987] Richard A. Bolt : The Integrated Multi-Modal Interface. *Transactions of the Institute of Electronics, Information, and Communication Engineers*, J70-D(11):2017–2025, novembre 1987.

[Bouchet, 2006] Jullien Bouchet (2006) : *Ingénierie de l'interaction multimodale en entrée Approche à composants ICARE*. Thèse de doctorat, Université Joseph Fourier, Grenoble, France, décembre 2006.

[Bouchet *et al.*, 2004] Jullien Bouchet, Laurence Nigay et Thierry Ganille : ICARE Software Components for Rapidly Developing Multimodal Interfaces. *In ICMI '04: Proceedings of the 6th International Conference on Multimodal Interfaces*, pages 251–258. ACM, 2004. doi:10.1145/1027933.1027975.

[Bouchet *et al.*, 2005] Jullien Bouchet, Laurence Nigay et Thierry Ganille : The ICARE Component-Based Approach for Multimodal Input Interaction: Application to Real-Time Military Aircraft Cockpits. *In HCI International '05: Proceedings of the 11th International Conference on Human-Computer Interaction*. L. Erlbaum Associates Inc., juillet 2005.

[Buxton, 1983] William Buxton : Lexical and Pragmatic Considerations of Input Structures. *SIGGRAPH Computer Graphics*, 17(1):31–37, janvier 1983. doi:10.1145/988584.988586.

[Callaú *et al.*, 2011] Oscar Callaú, Romain Robbes, Éric Tanter et David Röthlisberger : How Developers Use the Dynamic Features of Programming Languages: The Case of Smalltalk. *In MSR '11: Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 23–32. ACM, 2011. doi:10.1145/1985441.1985448.

[Card *et al.*, 1991] Stuart K. Card, Jock D. Mackinlay et George G. Robertson : A Morphological Analysis of the Design Space of Input Devices. *ACM Transactions on Information Systems (TOIS)*, 9(2):99–122, avril 1991. doi:10.1145/123078.128726.

[Cervantes et Hall, 2004] Humberto Cervantes et Richard S. Hall : Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. *In ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 614–623. IEEE Computer Society, mai 2004. doi:10.1109/ICSE.2004.1317483.

[Coutaz, 1987] Joëlle Coutaz : PAC, an Object Oriented Model for Dialog Design. *In Interact '87: Proceedings of the 2nd IFIP International Conference on Human-Computer Interaction*, volume 87, pages 431–436. septembre 1987.

[Coutaz, 2007] Joëlle Coutaz : Meta-User Interfaces for Ambient Spaces. *In* Karin Coninx, Kris Luyten, Kevin A. Schneider, Karin Coninx, Kris Luyten et KevinA. Schneider, éditeurs : *Task Models and Diagrams for Users Interface Design*, volume 4385 de *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-70816-2_1.

[Coutaz *et al.*, 2005] Joëlle Coutaz, James L. Crowley, Simon Dobson et David Garlan : Context Is Key. *Communications of the ACM*, 48(3):49–53, mars 2005. doi:10.1145/1047671.1047703.

[Dabrowski et Munson, 2001] James R. Dabrowski et Ethan V. Munson : Is 100 Milliseconds Too Fast? *In CHI EA '01: Extended Abstracts on Human Factors in Computing Systems*, pages 317–318. ACM, 2001. doi:10.1145/634067.634255.

[Dai, 2004] Guozhong Dai : Pen-Based User Interface. *In CACWD '04: Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design*, volume 2, pages I–32 – I–36. mai 2004. doi:10.1109/CACWD.2004.1349146.

[Dey, 2001] Anind K. Dey : Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, février 2001. 10.1007/s007790170019. doi:10.1007/s007790170019.

[Dragicevic, 2004] Pierre Dragicevic (2004) : *Un modèle d'interaction en entrée pour des systèmes interactifs multi-dispositifs hautement configurables*. Thèse de doctorat, Université de Nantes, France, mars 2004.

[Dragicevic et Fekete, 2004] Pierre Dragicevic et Jean-Daniel Fekete : Support for Input Adaptability in the ICon Toolkit. *In ICMI '04: Proceedings of the 6th International Conference on Multimodal Interfaces*, pages 212–219. ACM, 2004. doi:10.1145/1027933.1027969.

[Duchowski, 2002] Andrew T. Duchowski : A Breadth-First Survey of Eye-Tracking Applications. *Behavior Research Methods*, 34(4):455–470, novembre 2002. doi:10.3758/BF03195475.

[Dumas, 2010] Bruno Dumas (2010) : *Frameworks, Description Languages and Fusion Engines for Multimodal Interactive Systems*. Thèse de doctorat, University of Fribourg, Switzerland, décembre 2010.

[Dumas *et al.*, 2009] Bruno Dumas, Denis Lalanne et Sharon Oviatt : Multimodal Interfaces: A Survey of Principles, Models and Frameworks. *In* Denis Lalanne et Jürg Kohlas, éditeurs : *Human Machine Interaction*, volume 5440 de *Lecture Notes in Computer Science*, pages 3–26. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-00437-7_1.

[Endres *et al.*, 2005] Christoph Endres, Andreas Butz et Asa MacWilliams : A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. *Mobile Information Systems*, 1(1):41–80, janvier 2005.

[Escoffier, 2008] Clément Escoffier (2008) : *iPOJO : un modèle à composant à service flexible pour les systèmes dynamiques*. Thèse de doctorat, Université Joseph Fourier, Grenoble, France, décembre 2008.

[Escoffier *et al.*, 2007] Clément Escoffier, Richard S. Hall et Philippe Lalanda : iPOJO: An Extensible Service-Oriented Component Framework. *In SCC '07: Proceedings of the IEEE International Conference on Services Computing*, pages 474–481. IEEE Computer Society, juillet 2007. doi:10.1109/SCC.2007.74.

[Foley *et al.*, 1984] James D. Foley, Victor L. Wallace et Peggy Chan : The Human Factors of Computer Graphics Interaction Techniques. *IEEE Computer Graphics and Applications*, 4(11):13–48, novembre 1984.

[Gabillon *et al.*, 2011] Yoann Gabillon, Mathieu Petit, Gaëlle Calvary et Humbert Fiorino : Automated Planning for User Interface Composition. *In SEMAIS '11: Proceedings of the 2nd International Workshop on Semantic Models for Adaptive Interactive Systems, at IUI 2011 conference*. 2011.

[Garcia, 2012] Issac Garcia (2012) : *Modèles de conception et d'exécution pour la médiation et l'intégration de services*. Thèse de doctorat, Université de Grenoble, France, juin 2012.

[Garcia *et al.*, 2010] Issac Garcia, Gabriel Pedraza, Bassem Debbabi, Philippe Lalanda et Catherine Hamon : Towards a Service Mediation Framework for Dynamic Applications. *In APSCC '10: Services Computing Conference, IEEE Asia-Pacific*, pages 3–10. IEEE Computer Society, décembre 2010. doi:10.1109/APSCC.2010.90.

[Garlan *et al.*, 2002] David Garlan, Daniel P. Siewiorek, Asim Smailagic et Peter Steenkiste : Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, avril–juin 2002. doi:10.1109/MPRV.2002.1012334.

[Gosling *et al.*, 2005] James Gosling, Bill Joy, Guy Steele et Gilad Bracha : *The Java Language Specification*. Addison-Wesley, 3rd édition, 2005. isbn:978-0321246783.

[Greenberg *et al.*, 2011] Saul Greenberg, Nicolai Marquardt, Till Ballendat, Rob Diaz-Marino et Miaosen Wang : Proxemic Interactions: The New Ubicomp? *interactions*, 18(1): 42–50, janvier 2011. doi:10.1145/1897239.1897250.

[Honkola *et al.*, 2010] Jukka Honkola, Hannu Laine, Ronald Brown et Olli Tyrkko : Smart-M3 Information Sharing Platform. *In ISCC '10: Proceedings of the 2010 IEEE Symposium on Computers and Communications*, pages 1041–1046. IEEE Computer Society, juin 2010. doi:10.1109/ISCC.2010.5546642.

[Horvitz, 1999] Eric Horvitz : Principles of Mixed-initiative User Interfaces. *In CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 159–166. ACM, 1999. doi:10.1145/302979.303030.

[Johanson *et al.*, 2002] Brian Johanson, Armando Fox et Terry Winograd : The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):67–74, avril–juin 2002. doi:10.1109/MPRV.2002.1012339.

[Johnson *et al.*, 1993] Peter Johnson, Stephanie Wilson, Panos Markopoulos et James Pycock : ADEPT: Advanced Design Environment for Prototyping with Task Models. *In Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, page 56. ACM, 1993. doi:10.1145/169059.169074.

[Juang et Rabiner, 2004] Biing-Hwang Juang et Lawrence R. Rabiner : Automatic Speech Recognition – A Brief History of the Technology Development. Elsevier Encyclopedia of Language and Linguistics, 2004.

[Kakousis *et al.*, 2010] Konstantinos Kakousis, Nearchos Paspallis et George Angelos Papadopoulos : A Survey of Software Adaptation in Mobile and Ubiquitous Computing. *Enterprise Information Systems*, 4(4):355–389, novembre 2010. doi:10.1080/17517575.2010.509814.

[Kiczales *et al.*, 1997] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier et John Irwin : Aspect-Oriented Programming. *In* Mehmet Aksit et Satoshi Matsuoka, éditeurs : *ECOOP '97: Object-Oriented Programming*, volume 1241 de *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg, 1997. doi:10.1007/BFb0053381.

[Krasner et Pope, 1988] Glenn E. Krasner et Stephen T. Pope : A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[König, 2010] Werner A. König (2010) : *Design and Evaluation of Novel Input Devices and Interaction Techniques for Large, High-Resolution Displays*. Thèse de doctorat, University of Konstanz, Germany, septembre 2010.

[König *et al.*, 2010] Werner A. König, Roman Rädle et Harald Reiterer : Interactive Design of Multimodal User Interfaces. *Journal on Multimodal User Interfaces*, 3(3):197–213, avril 2010. doi:10.1007/s12193-010-0044-2.

[Lau et Wang, 2005] Kung-Kiu Lau et Zheng Wang : A Taxonomy of Software Component Models. *In SEAA '05: Proceedings of the 31st Euromicro Conference on Software Engineering and Advanced Applications*, pages 88–95. août–septembre 2005. doi:10.1109/EUROMICRO.2005.8.

[Lawson *et al.*, 2009] Jean-Yves L. Lawson, Ahmad-Amr Al-Akkad, Jean Vanderdonckt et Benoît Macq : An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-the-Shelf Heterogeneous Components. *In EICS '09: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 245–254. ACM, 2009. doi:10.1145/1570433.1570480.

[Lee, 2008] Johnny Chung Lee : Hacking the Nintendo Wii Remote. *IEEE Pervasive Computing*, 7(3):39–45, juillet–septembre 2008. doi:10.1109/MPRV.2008.53.

[Lorenz *et al.*, 2008] Andreas Lorenz, Markus Eisenhauer et Andreas Zimmermann : Elaborating a Framework for Open Human Computer Interaction with Ambient Services. *In PERMID '08: Proceedings of the 4th International Workshop on Pervasive Mobile Interaction Devices (Mobile Devices as Pervasive User Interfaces and Interaction Devices), in the Pervasive 2008 Workshop Proceedings*, pages 171–174. 2008.

[López-de Ipiña *et al.*, 2006] Diego López-de Ipiña, Juan Ignacio Vázquez, Daniel Garcia, Javier Fernández, Iván García, David Sáinz et Aitor Almeida : A Middleware for the Deployment of Ambient Intelligent Spaces. *In* Yang Cai et Julio Abascal, éditeurs : *Ambient Intelligence in Everyday Life*, volume 3864 de *Lecture Notes in Computer Science*, pages 239–255. Springer Berlin Heidelberg, 2006. doi:10.1007/11825890_12.

[Mackinlay *et al.*, 1990] Jock Mackinlay, Stuart K. Card et George G. Robertson : A Semantic Analysis of the Design Space of Input Devices. *Human-Computer Interaction*, 5(2–3):145–190, juin 1990.

[Mansoux *et al.*, 2007] Benoît Mansoux, Laurence Nigay et Jocelyne Troccaz : Output Multimodal Interaction: The Case of Augmented Surgery. *In* Nick Bryan-Kinns, Ann Blanford, Paul Curzon et Laurence Nigay, éditeurs : *People and Computers XX — Engage*, pages 177–192. Springer London, 2007. doi:10.1007/978-1-84628-664-3_14.

[Mitra et Acharya, 2007] Sushmita Mitra et Tinku Acharya : Gesture Recognition: A Survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 37(3):311–324, mai 2007. doi:10.1109/TSMCC.2007.893280.

[Moore, 2003] Melody M. Moore : Real-World Applications for Brain-Computer Interface Technology. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2):162–165, juin 2003. doi:10.1109/TNSRE.2003.814433.

[Morimoto et Mimica, 2005] Carlos Hitoshi Morimoto et Marcio R. M. Mimica : Eye Gaze Tracking Techniques for Interactive Applications. *Computer Vision and Image Understanding*, 98(1):4–24, avril 2005. doi:10.1016/j.cviu.2004.07.010.

[Newman, 1968] William M. Newman : A System for Interactive Graphical Programming. *In AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, pages 47–54. ACM, 1968. doi:10.1145/1468075.1468083.

[Niezen *et al.*, 2010] Gerrit Niezen, Bram J. J. van der Vlist, Jun Hu et Loe M. G. Feijs : From Events to Goals: Supporting Semantic Interaction in Smart Environments. *In ISCC '10: Proceedings of the 2010 IEEE Symposium on Computers and Communications*, pages 1029–1034. IEEE Computer Society, juin 2010. doi:10.1109/ISCC.2010.5546634.

[Niezen *et al.*, 2012] Gerrit Niezen, Bram J. J. van der Vlist, Jun Hu et Loe M. G. Feijs : Using Semantic Transformers to Enable Interoperability Between Media Devices in a Ubiquitous Computing Environment. *In* Mika Rautiainen, Timo Korhonen, Edward Mutafungwa, Eila Ovaska, Artem Katasonov, Antti Evesti, Heikki Ailisto, Aaron Quigley,

Jonna Häkkilä, Natasa Milic-Frayling et Jukka Riekki, éditeurs : *Grid and Pervasive Computing Workshops*, volume 7096 de *Lecture Notes in Computer Science*, pages 44–53. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-27916-4_6.

[Nigay et Coutaz, 1996] Laurence Nigay et Joëlle Coutaz : Espaces conceptuels pour l'interaction multimédia et multimodale. *Technique et Science Informatique (TSI), spécial Multimédia et Collecticiel*, 15(9):1195–1225, 1996.

[Nigay et Coutaz, 1997] Laurence Nigay et Joëlle Coutaz : Multifeature Systems: The CARE Properties and Their Impact on Software Design. *In* John Lee, éditeur : *Intelligence and Multimodality in Multimedia Interfaces: Research and Applications*, chapitre 9. AAAI Press, 1997.

[Norman, 2010] Donald A. Norman : Natural User Interfaces are Not Natural. *interactions*, 17(3):6–10, mai 2010. doi:10.1145/1744161.1744163.

[osgi43, 2011] *OSGi Service Platform Core Specification*. The OSGi Alliance, avril 2011. Release 4, version 4.3. `http://www.osgi.org/Download/Release4V43`.

[Oviatt, 1997] Sharon Oviatt : Multimodal Interactive Maps: Designing for Human Performance. *Human-Computer Interaction*, 12(1–2):93–129, mars 1997.

[Oviatt, 1999] Sharon Oviatt : Ten Myths of Multimodal Interaction. *Communications of the ACM*, 42(11):74–81, novembre 1999. doi:10.1145/319382.319398.

[Papazoglou et Georgakopoulos, 2003] Mike P. Papazoglou et Dimitrios G. Georgakopoulos : Service-Oriented Computing. *Communications of the ACM*, 46(10):24–28, octobre 2003. doi:10.1145/944217.944233.

[Peterson *et al.*, 2006] Brian S. Peterson, Rusty O. Baldwin et Jeffrey P. Kharoufeh : Bluetooth Inquiry Time Characterization and Selection. *IEEE Transactions on Mobile Computing*, 5(9):1173–1187, septembre 2006. doi:10.1109/TMC.2006.125.

[Ponnekanti *et al.*, 2001] Shankar R. Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan et Terry Winograd : ICrafter: A Service Framework for Ubiquitous Computing Environments. *In* Gregory Abowd, Barry Brumitt et Steven Shafer, éditeurs : *Ubicomp 2001: Ubiquitous Computing*, volume 2201 de *Lecture Notes in Computer Science*, pages 56–75. Springer Berlin Heidelberg, 2001. doi:10.1007/3-540-45427-6_7.

[Primi, 2007] Marco Primi : The Erlang Programming Language. 2007. Programming Languages 2007 Report.

[Román *et al.*, 2002] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell et Klara Nahrstedt : A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, octobre 2002. doi:10.1109/MPRV.2002.1158281.

[Satyanarayanan, 2001] Mahadev Satyanarayanan : Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, août 2001. doi:10.1109/98.943998.

[Schilit *et al.*, 1994] Bill N. Schilit, Norman Adams et Roy Want : Context-Aware Computing Applications. *In WMCSA '94: Proceedings of the 1st Workshop on Mobile Computing Systems and Applications, 1994.* ., pages 85–90. décembre 1994. doi:10.1109/WMCSA.1994.16.

[Scoditti *et al.*, 2011] Adriano Scoditti, Renaud Blanch et Joëlle Coutaz : A Novel Taxonomy for Gestural Interaction Techniques Based on Accelerometers. *In IUI '11: Proceedings of the 15th International Conference on Intelligent User Interfaces*, pages 63–72. ACM, 2011. doi:10.1145/1943403.1943414.

[Serrano, 2010] Marcos Serrano (2010) : *Interaction multimodale en entrée : Conception et Prototypage*. Thèse de doctorat, Université de Grenoble, France, juin 2010.

[Serrano *et al.*, 2008a] Marcos SERRANO, David JURAS et Laurence NIGAY : A Three-Dimensional Characterization Space of Software Components for Rapidly Developing Multimodal Interfaces. *In ICMI '08: Proceedings of the 10th International Conference on Multimodal Interfaces*, pages 149–156, Chania, Crete, Greece. ACM, 2008a. doi:10.1145/1452392.1452421.

[Serrano et Nigay, 2009] Marcos SERRANO et Laurence NIGAY : Temporal Aspects of CARE-Based Multimodal Fusion: From a Fusion Mechanism to Composition Components and WoZ Components. *In ICMI-MLMI '09: Proceedings of the 2009 International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*, pages 177–184. ACM, 2009. doi:10.1145/1647314.1647346.

[Serrano *et al.*, 2008b] Marcos SERRANO, Laurence NIGAY, Jean-Yves L. LAWSON, Andrew RAMSAY, Roderick MURRAY-SMITH et Sebastian DENEF : The OpenInterface Framework: A Tool for Multimodal Interaction. *In CHI EA '08: Extended Abstracts on Human Factors in Computing Systems*, pages 3501–3506. ACM, 2008b. doi:10.1145/1358628.1358881.

[Sutherland, 1964] Ivan E. SUTHERLAND : Sketchpad: A Man-Machine Graphical Communication System. *In DAC '64: Proceedings of the SHARE Design Automation Workshop*, pages 6:329–6:346. ACM, 1964. doi:10.1145/800265.810742.

[Sutherland, 1968] Ivan E. SUTHERLAND : A Head-Mounted Three Dimensional Display. *In AFIPS '68 (Fall, part I): Proceedings of the December 9–11, 1968, Fall Joint Computer Conference*, pages 757–764. ACM, 1968. doi:10.1145/1476589.1476686.

[Szyperski *et al.*, 2002] Clemens SZYPERSKI, Dominik GRUNTZ et Stephan MURER : *Component Software: Beyond Object-Oriented Programming.* ACM Press/Addison-Wesley, 2nd édition, 2002. isbn:978-0201745726.

[Thevenin et Coutaz, 1999] David THEVENIN et Joëlle COUTAZ : Plasticity of User Interfaces: Framework and Research Agenda. *In Interact '99: Proceedings of the 7th IFIP International Conference on Human-Computer Interaction*, volume 99, pages 110–117, Edinburgh, Scotland. Chapman & Hall, 1999.

[Tigli *et al.*, 2009] Jean-Yves TIGLI, Stéphane LAVIROTTE, Gaëtan REY, Vincent HOURDIN, Daniel CHEUNG-FOO-WO, Eric CALLEGARI et Michel RIVEILL : WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-Based Web Services. *Annals of Telecommunications*, 64(3–4):197–214, avril 2009. doi:10.1007/s12243-008-0081-y.

[uml241, 2011] *OMG Unified Modeling LanguageTM (OMG UML), Superstructure.* Object Management Group, août 2011. Version 2.4.1. `http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/`.

[United Nations Environment Programme, 2005] DEWA / GRID-Europe UNITED NATIONS ENVIRONMENT PROGRAMME : E-waste, the Hidden Side of IT Equipment's Manufacturing and Use. Environment Alert Bulletin, janvier 2005.

[upnp11, 2008] *UPnP Device Architecture.* UPnP Forum, octobre 2008. Version 1.1. `http://www.upnp.org/resources/upnpresources.zip`.

[usb20, 2000] *Universal Serial Bus Specification*, avril 2000. Version 2.0. `http://www.usb.org/developers/docs/`.

[usbhid111, 2001] *Device Class Definition for Human Interface Devices (HID).* USB Implementers' Forum, juin 2001. Version 1.11. `http://www.usb.org/developers/devclass_docs/HID1_11.pdf`.

[usbhid112, 2004] *HID Usage Tables.* USB Implementers' Forum, octobre 2004. Version 1.12. `http://www.usb.org/developers/devclass_docs/Hut1_12v2.pdf`.

[van der Vlist *et al.*, 2010] Bram J. J. van der Vlist, Gerrit Niezen, Jun Hu et Loe M. G. Feijs : Semantic Connections: Exploring and Manipulating Connections in Smart Spaces. *In ISCC '10: Proceedings of the 2010 IEEE Symposium on Computers and Communications*, pages 1–4. IEEE Computer Society, juin 2010. doi:10.1109/ISCC.2010.5546636.

[Vanderhulst *et al.*, 2008] Geert Vanderhulst, Kris Luyten et Karin Coninx : ReWiRe: Creating Interactive Pervasive Systems that Cope with Changing Environments by Rewiring. *In IE '08: Proceedings of the IET 4th International Conference on Intelligent Environments*, pages 1–8. juillet 2008.

[Vicaria *et al.*, 2008] Juan A. Vicaria, José M. Maestre et Eduardo F. Camacho : Academical and Research Wiimote Applications. *In* Katherine Blashki, éditeur : *Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction*. juillet 2008.

[Vybornova *et al.*, 2008] Olga Vybornova, Hildeberto Mendonça, Jean-Yves L. Lawson et Benoît Macq : High Level Data Fusion on a Multimodal Interactive Applications Platform. *In ISM '08: Proceedings of the 10th IEEE International Symposium on Multimedia*, pages 493–494. IEEE Computer Society, 2008. doi:10.1109/ISM.2008.21.

[Wallace, 1976] Victor L. Wallace : The Semantics of Graphic Input Devices. *In The Papers of the ACM Symposium on Graphic Languages*, pages 61–65. ACM, 1976. doi:10.1145/800143.804734.

[Weiser, 1991] Mark Weiser : The Computer for the 21st Century. *Scientific American*, 265(3): 94–104, septembre 1991. doi:10.1038/scientificamerican0991-94.

[Wiederhold, 1992] Gio Wiederhold : Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, mars 1992. doi:10.1109/2.121508.

[Yang, 2003] Jian Yang : Web Service Componentization. *Communications of the ACM*, 46(10): 35–40, octobre 2003. doi:10.1145/944217.944235.

[Zhu *et al.*, 2005] Feng Zhu, Matt W. Mutka et Lionel M. Ni : Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, 4(4):81–90, octobre–décembre 2005. doi:10.1109/MPRV.2005.87.