Apprentissage par renforcement de modeles de contexte pour l'informatique ambiante

Sofia Zaidenberg

Grenoble INP

THÈSE

pour obtenir le grade de

Docteur Du Groupe Grenoble INP

Spécialité : Informatique

préparée au Laboratoire d'Informatique de Grenoble

dans le cadre de l'École Doctorale Mathématiques,
Sciences et Technologies de l'Information, Informatique

présentée et soutenue publiquement

par

Sofia Zaidenberg

le 16 octobre 2009

Apprentissage par renforcement de modèles de contexte pour l'informatique ambiante

---

Directeur de thèse : James L. Crowley
Co-directeur de thèse : Patrick Reignier

JURY

M^me^ Brigitte Plateau	Présidente
M. Olivier Sigaud	Rapporteur
M. Olivier Boissier	Rapporteur
M. James L. Crowley	Directeur de thèse
M. Patrick Reignier	Co-directeur de thèse
M^me^ Marie-Pierre Gleizes	Examinatrice

Résumé

Cette thèse étudie l'acquisition automatique par apprentissage d'un modèle de contexte pour un utilisateur dans un environnement ubiquitaire. Dans un tel environnement, les dispositifs peuvent communiquer et coopérer afin de former un espace informatique cohérent. Certains appareils ont des capacités de perception, utilisées par l'environnement pour détecter la situation – le *contexte* – de l'utilisateur. D'autres appareils sont capables d'exécuter des actions. La problématique que nous nous sommes posée est de déterminer les associations optimales pour un utilisateur donné entre les situations et les actions. L'*apprentissage* apparaît comme une bonne approche car il permet de personnaliser l'environnement sans spécification explicite de la part de l'usager. Un apprentissage à vie permet, par ailleurs, de toujours s'adapter aux modifications du monde et des préférences utilisateur. L'apprentissage par renforcement est un paradigme d'apprentissage qui peut être une solution à notre problème, à condition de l'adapter aux contraintes liées à notre cadre d'application.

Abstract

This thesis studies the automatic acquisition by machine learning of a context model for a user in a ubiquitous environment. In such an environment, devices can communicate and cooperate in order to create a consistent computerized space. Some devices possess perceptual capabilities. The environment uses them to detect the user's situation – his *context*. Other devices are able to execute actions. Our problematics consists in determining the optimal associations, for a given user, between situations and actions. *Machine learning* seems to be a sound approach since it results in a customized environment without requiring an explicit specification from the user. A life long learning lets the environment adapt itself continuously to world changes and user preferences changes. Reinforcement learning can be a solution to this problem, as long as it is adapted to some particular constraints due to our application setting.

Remerciements

Avant tout, j'aimerais remercier mes encadrants de thèse, mon co-directeur Dr. Patrick Reignier et mon directeur Prof. James L. Crowley, pour m'avoir donné l'opportunité d'entrer dans le monde de la recherche. Je leur suis reconnaissante de m'avoir guidée, écoutée et conseillée tout au long de ces quatre années de thèse. Je remercie particulièrement Nadine Mandran pour son expertise et pour avoir dirigé une enquête grand public qui a grandement aidé ma thèse. Je remercie également Prof. Olivier Sigaud, Prof. Olivier Boissier, Prof. Marie-Pierre Gleizes et Prof. Brigitte Plateau pour leur intérêt dans mon travail et pour avoir accepté de faire partie du jury de cette thèse.

Puis, j'aimerais remercier tous les membres de l'équipe Prima pour leur soutien durant ces années et pour m'avoir permis de travailler dans une ambiance dynamique. En particulier, je suis reconnaissante à mes collègues Rémi Émonet et Rémi Barraquand pour les nombreuses discussions fructueuses pour la recherche, et à Matthieu Langet pour son intervention technique dans mon travail.

Par ailleurs, je souhaite adresser un merci aux personnes m'ayant permis de découvrir le monde de l'enseignement durant ma thèse, je pense en particulier à mon encadrant Patrick Reignier, à Philippe Genoud et à Augustin Lux, qui m'ont donné l'opportunité d'enseigner et de découvrir la direction à prendre suite à cette thèse.

Enfin, je remercie mes parents et mon entourage pour leur soutien et leurs encouragements durant cette thèse.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Introduction

Nos environnements et nos vies sont remplis d'une multitude d'appareils informatiques qui se contentent de jouer chacun leur rôle indépendamment les uns des autres et inconscients du monde qui les entoure. L'informatique ambiante vise à les orchestrer pour constituer un tout cohérent, un réseau d'appareils interconnectés et communiquants, jouant désormais leur rôle dans une pièce dont ils sont tous acteurs. Le but final de cette comédie est de tapisser l'environnement de l'utilisateur d'une couche transparente d'intelligence, capable de le comprendre et de lui être utile partout et tout le temps, sans le perturber. Les dispositifs unitaires deviennent alors des parties d'un ordinateur virtuel, intégré à l'environnement et à notre vie quotidienne, interagissant avec l'utilisateur d'une manière naturelle, presque en arrière-plan de son attention.

1.1 L'informatique ubiquitaire

Le domaine de l'informatique ubiquitaire a été introduit au début des années 1990 par Mark Weiser au laboratoire Xerox PARC [Weiser, 1991]. La vision de Weiser a depuis motivé de nombreux chercheurs et industriels. Cette vision dépeint une technologie omniprésente mais calme, transportant de l'information, toujours à disposition de l'utilisateur sans explicitement attirer son attention. Une première partie de cette vision, qui constitue un domaine de recherche en soi, est la conception de systèmes sensibles au contexte *(context-aware computing)*. Ces recherches s'intéressent à définir, modéliser et détecter le contexte d'un utilisateur dans l'environnement.

Avec l'introduction de la notion de contexte s'est progressivement dégagée la notion d'intelligence ambiante. Celle-ci a pour idée centrale d'utiliser le système ubiquitaire pour acquérir le contexte de l'utilisateur et de s'y adapter dynamiquement. L'intelligence provient de l'adéquation du système et des services qu'il propose à la situation courante de l'utilisateur.

Tenir compte du contexte est un premier pas vers l'informatique *calme* de Weiser. Ceci permet aux appareils de ne pas être mal à propos. Dans les dernières années, ces notions ont quelque peu évolué. La faculté à s'adapter à l'utilisateur et à son état reste un élément central dans la conception des systèmes. Mais le problème rencontré était celui de la pertinence des systèmes et des services qu'ils pouvaient rendre. Les recherches étaient tournées vers la création de sys-

tèmes complexes d'intelligence artificielle, oubliant quelque fois l'utilité vis-à-vis de l'utilisateur. La nouvelle idée apparue vers 2005 est de faire des systèmes ubiquitaires permettant aux humains de trouver leurs propres usages des outils proposés.

1.2 Acquisition automatique du modèle de contexte

L'intelligence ambiante propose de se placer dans un système ubiquitaire, de détecter le contexte de l'utilisateur et de fournir des services dépendants de la situation courante des usagers. La question qui se pose alors est de trouver les associations entre les situations et les services rendus. Ces correspondances constituent ce que nous appelons le *modèle de contexte*. Le cadre d'application étant l'informatique ubiquitaire, les situations doivent décrire, d'une manière suffisamment précise pour être pertinente, une partie du monde réel, avec toute sa complexité et son dynamisme. Un système véritablement adapté à l'utilisateur choisit un service en fonction du contexte de ce dernier, mais également en fonction de l'utilisateur lui-même. Dans une même situation, différentes personnes ont différentes préférences quant aux actions du système. Comment alors choisir ces associations entre contexte et services de manière personnalisée à chaque utilisateur ?

Faire spécifier le modèle de contexte par un expert ne permet pas de le rendre adapté à chaque utilisateur, ni de le faire évoluer avec l'environnement ou avec les changements dans les préférences de l'utilisateur. Le faire spécifier par l'utilisateur est une tâche trop lourde et va à l'encontre de l'idée d'un système qui sait se faire oublier. Par contre, laisser à l'utilisateur le choix d'en spécifier une partie permet de l'engager et de lui donner un sentiment d'implication dans le système. Il ne faut toutefois pas attendre une granularité fine dans la partie du modèle définie par l'usager.

1.3 Apprentissage par renforcement du modèle de contexte

Il reste une possibilité pour trouver les actions adéquates à chaque situation : le système doit déterminer le modèle de contexte par lui-même. À partir des interactions avec l'utilisateur et de l'observation de ses réactions aux actions du système, celui-ci doit trouver le comportement qui satisfait au maximum l'utilisateur. Le système ne doit jamais baisser sa vigilance car l'utilisateur change d'avis, évolue ses préférences et modifie son environnement. Cette thèse a pour but d'étudier la construction automatique du modèle de contexte dans un environnement ubiquitaire intelligent.

Cette adaptation automatique s'obtient par *apprentissage*. La technique d'apprentissage automatique choisie devra respecter plusieurs contraintes. Entre autres, l'entraînement doit être facile car il sera à la charge de l'utilisateur. L'apprentissage doit être à vie à cause des évolutions mentionnées plus haut. Il doit laisser la possibilité d'injecter des connaissances initiales car dès son installation, le système doit avoir un minimum de cohérence dans son comportement.

L'apprentissage doit fournir un résultat rapide car l'utilisateur risque de se lasser d'entraîner un système sans constater l'intégration des entrées de son entraînement. Enfin, la contrainte probablement la plus importante est l'*intelligibilité* du système pour l'utilisateur.

Un système ambiant est complexe et peut être doté de nombreuses capacités en matière de services rendus. Puisque son comportement est appris progressivement et non pas prédéfini, il évolue en permanence. L'utilisateur doit ainsi comprendre et s'adapter à un système complexe et évolutif, qui, de plus, peut faire des erreurs. Par conséquent, il est important également de gagner la confiance de l'utilisateur. Pour faciliter ceci, le système doit être capable de *s'expliquer* à l'utilisateur. Les représentations internes du système doivent être transparentes à l'utilisateur.

Le paradigme d'apprentissage que nous avons choisi est l'*apprentissage par renforcement*. Il peut, en effet, répondre à tous ces critères, à condition de l'adapter à notre cadre d'application spécifique. L'entraînement demande très peu d'efforts à l'utilisateur : il lui suffit de donner une appréciation subjective, positive ou négative, du comportement du système. Cette récompense peut également être recueillie implicitement, à partir d'indices. L'apprentissage par renforcement classique fonctionne par essais-erreurs. Pour nous, ceci est un problème car les erreurs impliquent directement l'utilisateur et il faut donc les limiter au maximum.

L'apprentissage par renforcement *indirect* est une technique pour accélérer l'apprentissage dans le monde réel par l'utilisation d'un monde virtuel. La plupart des essais se font dans ce monde simulé. Mais comment simuler l'environnement réel ? Nous proposons un modèle du monde et des récompenses utilisateur, et deux algorithmes d'apprentissage supervisé permettant d'acquérir ce modèle à partir d'exemples du monde réel.

L'initialisation de ce modèle nous permet également de fournir un comportement initial au système, par une première phase d'apprentissage se déroulant uniquement dans le monde virtuel. Le problème de la complexité de l'environnement est abordé par des techniques de généralisation du comportement et du modèle. Enfin, l'intelligibilité est atteinte par la modélisation du problème. L'apprentissage par renforcement s'applique sur un ensemble d'états et d'actions. Il n'y a pas de contraintes sur la définition de ces ensembles. Par conséquent, nous avons choisi des représentations humainement lisibles. À tout moment, le système peut montrer son état à l'utilisateur, ce qui le rend transparent.

1.4 Expérimentations

La mise en pratique de la technique d'apprentissage proposée nécessite la mise en œuvre d'un système d'informatique ubiquitaire. Nous avons réalisé un tel système. Notre architecture est distribuée et composée de modules interconnectés et communiquants. Nous distinguons en particulier les modules capteurs, utilisés pour détecter le contexte de l'utilisateur, et les modules effecteurs, permettant d'exécuter des actions dans l'environnement et de rendre des services à l'utilisateur. Le système est personnifié par un assistant personnel virtuel qui orchestre les autres modules. Cet assistant possède une base de connaissances sur l'environnement, les utilisateurs et les services disponibles. Il enregistre également un historique de tous les événements concernant l'utilisateur, le contexte

ou les services. Cette base lui sert d'une part pour l'apprentissage supervisé du modèle du monde, et d'autre part pour pouvoir fournir des explications a posteriori à l'utilisateur. Une telle architecture doit être dynamique, elle doit pouvoir intégrer des dispositifs nomades, entrant et quittant l'environnement sans prévenir (ce qui est notamment le cas des dispositifs portables de l'utilisateur, tels que son téléphone cellulaire ou son PDA). Les services rendus utilisent potentiellement tous les appareils disponibles, afin d'exécuter le service de la manière la plus appropriée. L'assistant doit pouvoir contrôler le cycle de vie des modules à distance et dynamiquement.

Dans cet environnement rendu ubiquitaire, nous avons mis en place nos expériences. Les premières expérimentations ont, cependant, utilisé un simulateur de l'environnement, afin de simplifier les tests et pouvoir contrôler les conditions d'expérimentation. Ces expérimentations sont quantitatives, elles mesurent les performances de nos algorithmes d'apprentissage. L'aspect qualitatif a été mesuré par une enquête réalisée auprès d'utilisateurs potentiels en leur présentant une maquette de l'assistant.

1.5 Résultats

L'évaluation des résultats consiste à déterminer à quel point le comportement appris de l'assistant correspond à ce que désire l'utilisateur. Il s'agit de compter le nombre de situations dans lesquelles l'action de l'assistant est la bonne pour l'utilisateur. Les résultats montrent qu'au cours de l'apprentissage, ce nombre augmente. L'assistant satisfait donc de plus en plus l'utilisateur. Il s'adapte rapidement aux nouvelles situations.

L'enquête auprès d'utilisateurs potentiels a montré qu'une partie non négligeable des sujets serait intéressée par un tel système. Nous avons également eu la confirmation que les utilisateurs ont, en effet, besoin de comprendre l'assistant, d'en obtenir des explications, afin de l'accepter et de mieux tolérer ses erreurs. Les utilisateurs ne souhaitent pas être dérangés ou trop sollicités par le système, mais un bon nombre souhaite également avoir la possibilité d'expliciter leurs préférences concernant les services.

1.6 Structure du manuscrit

Ce manuscrit est organisé comme suit.

Le chapitre 2 présente le problème en détail en commençant par une étude de l'état de l'art depuis la naissance de l'informatique ubiquitaire jusqu'à aujourd'hui, en passant par la vision actuelle du futur du domaine. Au vu de cet état de l'art, et après avoir présenté l'objectif de cette thèse, ce chapitre dégage les contraintes à respecter par la méthode proposée.

Le chapitre 3 détaille l'enquête que nous avons menée auprès d'utilisateurs potentiels. Il présente le choix des sujets dans différentes catégories, la grille d'entretien utilisée lors des interviews et la maquette présentée aux sujets. Enfin, il présente les idées dégagées par les personnes interrogées, leurs réactions et questions soulevées. Nous verrons que les sujets ont confirmé la plupart de nos hypothèses mais nous ont également apporté des éléments inattendus et intéressants.

Le chapitre 4 est consacré à la réalisation d'un système ambiant. Il explique les besoins d'un tel système et les technologies utilisées pour le réaliser en remplissant ces besoins. Il présente la base de données servant de connaissance et de mémoire à l'assistant. Enfin, ce chapitre introduit les modules capteurs et effecteurs réalisés, et la manière dont l'assistant interagit avec ceux-ci.

Le chapitre 5 est dédié à l'apprentissage par renforcement. Il commence par une présentation formelle de ce paradigme. Puis, il présente un état de l'art de systèmes similaires au nôtre utilisant également l'apprentissage par renforcement. Ce chapitre expose également la technique d'apprentissage par renforcement indirect. Ensuite, il formule la manière dont nous avons appliqué cette technique. Nous décrivons notre représentation des situations de l'environnement, la manière dont nous les détectons en utilisant nos modules capteurs, et la manière dont nous traitons cette information pour rendre des services à l'utilisateur. Nous nous intéressons ensuite à notre formulation du modèle du monde et aux algorithmes d'apprentissage supervisé qui nous permettent de l'acquérir. Enfin, nous récapitulons le fonctionnement global de notre assistant et la dynamique commune des éléments le constituant.

Avant de conclure ce manuscrit dans le chapitre 8, le chapitre 6 présente les expérimentations que nous avons effectuées. Celles-ci visent à tester l'adéquation du comportement appris aux souhaits de l'utilisateur. Les premières expériences utilisent un simulateur de l'environnement et jouent un scénario prédéfini. Dans la suite, un expérimentateur génère des interactions servant à apprendre le modèle du monde, utilisé ensuite pour l'apprentissage par renforcement. Les résultats montrent le succès de l'apprentissage. Le chapitre 7, lui, présente les résultats de ces expériences, non sans avoir au préalable décrit notre manière de mesurer un résultat.

Chapitre 2

Présentation du problème à résoudre

2.1 Introduction à l'informatique ubiquitaire

Le terme « informatique ubiquitaire » est équivalent au terme « informatique ambiante » et vient de l'anglais *ubiquitous computing* ; le terme communément employé en est le raccourci : *ubicomp*, terme qui pourra être rencontré au long de ce manuscrit. Dans la vision classique de l'ubicomp, l'informatique sort de l'ordinateur pour s'intégrer directement dans l'environnement, qui devient donc ubiquitaire. Le but de cette idée est d'élargir les possibilités de l'informatique, de faire en sorte que l'ordinateur profite à l'utilisateur à tout moment lorsqu'il se trouve dans cet environnement. Pour aller plus loin, cet ordinateur ambiant offre des capacités d'interaction plus naturelles, ce qui le rend transparent et utilisable sans effort pour les personnes. Un exemple d'un tel environnement est représenté par la figure 2.1 où l'on peut voir un bureau équipé de divers appareils (ordinateurs fixes ① et portables ②, téléphones portables ③, haut-parleurs ④, vidéo-projecteurs ⑤, caméras ⑥, murs de microphones ⑦, antennes WiFi, Bluetooth et RFiD ⑧, etc.).

De même que l'intelligence artificielle tente d'augmenter les capacités d'un ordinateur, l'informatique ambiante peut s'augmenter en l'intelligence ambiante. Le système défini ci-dessus se voit ajouter des capacités de perception et d'analyse de l'environnement, des utilisateurs et de leur activité, ce qui lui permet de réagir en fonction du *contexte* (dont il est question section 2.2.5). Pour ceci, il est nécessaire de formaliser et structurer les informations perçues par les différents capteurs qui vont équiper cet environnement. De cette façon, ces dispositifs divers vont travailler ensemble, de façon coordonnée, pour atteindre un but commun. La figure 2.2 illustre cette idée de réseau de dispositifs hétérogènes.

Afin de concrètement réaliser ce projet, partir sur l'idée d'intégrer uniquement des dispositifs adaptés à notre tâche est voué à l'échec. En effet, les environnements (entreprises, lieux publics et habitats) sont déjà largement équipés en matériel informatique et n'accepteront pas de tout remplacer par des appareils dédiés. Par conséquent, il est préférable d'utiliser les dispositifs déjà présents et disponibles dans l'environnement. Nous travaillons avec tous les appareils que nous avons « sous la main » : les ordinateurs bien sûr, les téléphones portables,

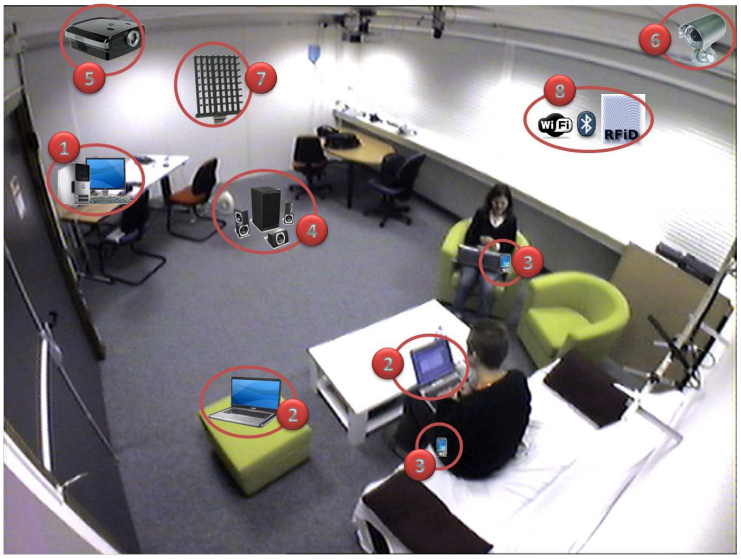

FIG. 2.1 – Un exemple de « bureau ubiquitaire » (le *SmartOffice* de l'équipe PRIMA) : un monde où l'utilisateur est entouré de divers appareils avec lesquels il interagit et qui communiquent également entre eux.

FIG. 2.2 – Illustration de l'informatique ubiquitaire intégrant de nombreux appareils hétérogènes dans le but de les faire collaborer.

les PDAs, les caméras, les microphones, les haut-parleurs, les projecteurs, les imprimantes, etc. Tous ces appareils sont déjà présents et il n'était pas prévu initialement de les utiliser dans un contexte d'environnement intelligent. Il faut trouver un moyen de faire communiquer ces appareils divers, de pouvoir recueillir des informations qu'ils peuvent fournir ou bien les piloter afin d'utiliser leurs fonctionalités pour rendre des services à l'utilisateur. De plus, notre système ambiant doit également répondre à la contrainte d'être facile à maintenir. Cette problématique sera abordée dans la section 4.2.

Une fois ce problème résolu, nous disposons d'un ensemble de capteurs, injectant des informations dans une base commune sensée refléter le contexte, ou bien l'état de l'environnement. Nous allons nous reposer sur cet état perçu pour proposer automatiquement des services. Ces services sont des actions que le système décide d'exécuter en utilisant l'ensemble des dispositifs faisant partie de l'environnement ambiant. Notre but est de faire en sorte que ces services soient personnalisés pour chaque utilisateur, de la même manière qu'il peut personnaliser son ordinateur (fond d'écran, raccourcis clavier, couleurs et polices, etc.).

Mais il s'agit ici d'un système bien plus complexe qu'un bureau informatique. De plus, le système devra être fait en sorte de pouvoir évoluer facilement. Par exemple, de nouveaux dispositifs intégreront sans doute fréquemment l'environnement ; celui-ci deviendra de plus en plus complexe. En outre, il est hautement probable que les préférences de l'utilisateur changent dans le temps. Pour ces raisons, nous ne voulons pas demander à l'utilisateur de spécifier lui-même ses préférences. Cette tâche est trop complexe et devra être répétée à intervalles de temps réguliers. Il faut *apprendre* ces préférences. Qui plus est, cet apprentissage est à vie *(life long learning)* à cause des évolutions constantes mentionnées ci-dessus.

Dans la suite de ce chapitre, nous allons présenter le domaine de l'informatique ubiquitaire, depuis sa naissance au début des années 1990, jusqu'à aujourd'hui (section 2.2). Au vu de l'état de l'art établi, nous donnerons une présentation avisée de notre objectif (section 2.3), et des difficultés à résoudre afin de l'atteindre (section 2.4). Enfin, nous illustrerons notre objectif sur un scénario exemple (section 2.5), décrit ici d'une manière grossière et qui sera repris au long de ce manuscrit en y ajoutant de plus en plus de détails.

2.2 État de l'art

Nous allons présenter l'état de l'art en informatique ubiquitaire sous forme chronologique, en montrant les racines du domaine et son évolution.

- 1991 : l'informatique ubiquitaire est introduite par Mark Weiser [Weiser, 1991] ;
- 1999 : Philips [Weber *et al.*, 2005] et l'ISTAG [Ducatel *et al.*, 2001] développent la notion d'intelligence ambiante, basée sur l'informatique ubiquitaire ;
- 2005 : un constat est fait sur l'avancement de l'informatique ubiquitaire par rapport à la vision initiale de Weiser [Bell et Dourish, 2007, Rogers, 2006] ;
- 2008 : la vision du futur (notamment par Microsoft [Harper *et al.*, 2008]).

2.2.1 1991 – La vision de Weiser

L'informatique ubiquitaire s'inspire de nombreux aspects de l'informatique, mais a été introduite en tant que telle par Mark Weiser en 1988 dans le centre de recherche *Computer Science Lab* de Xerox PARC. Dans son article fondateur [Weiser, 1991], Weiser soutient :

> « The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. »

Ainsi, pour Weiser, les technologies les plus profondes sont celles qui disparaissent. Elles se faufilent dans la structure de la vie quotidienne jusqu'à en devenir indiscernables. Pour réaliser cela, trois éléments sont nécessaires : des ordinateurs bon marché et à faible consommation électrique qui proposent des surfaces d'affichage commodes, des logiciels d'informatique ubiquitaire et un réseau qui lie tous les appareils. L'informatique ubiquitaire ne vit pas dans un dispositif particulier, mais est imprégnée dans la structure même de l'environnement qui nous entoure.

Weiser part du constat que bientôt, chaque personne sera entourée par de nombreux ordinateurs, comme l'illustre la figure 2.3. L'informatique qui a pour but d'utiliser tous ces ordinateurs ensemble est ce qu'il appelle l'informatique ubiquitaire. Mais pour lui, cet ordinateur du futur doit être invisible. Il ne doit pas s'imposer à notre conscience, il doit agir de manière analogue à des lunettes : elles jouent leur rôle sans encombrer notre esprit. Mark Weiser argumente qu'un bon outil est présent en tâche de fond, contrairement à ce que l'on peut voir dans la science fiction où les outils attirent l'attention et sont captivants. Il propose la métaphore de notre enfance à l'informatique ubiquitaire : une fondation invisible, rapidement oubliée mais qui nous accompagne en permanence et qui est utilisée sans effort [Weiser, 1994].

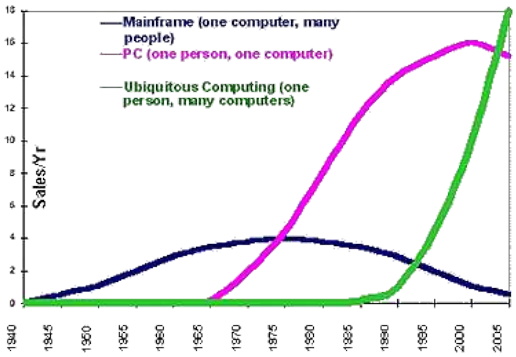

FIG. 2.3 – Les grandes tendances en informatique ubiquitaire. Source : http://sandbox.xerox.com/ubicomp/.

L'incarnation initiale de l'informatique ubiquitaire a pris la forme de trois types de dispositifs : les *tabs*, *pads* et *boards* développés à Xerox PARC entre 1988 et 1994. Les *tabs* [Adams *et al.*, 1993] sont les dispositifs les plus petits des trois. Ils tiennent dans la main et sont des PDAs qui peuvent communiquer par infrarouge avec un réseau d'émetteurs-récepteurs afin d'accéder à des applications

s'exécutant sur des stations de travail classiques. Les *pads* ont une taille plus grande et leur utilisation peut être comparée à celle d'une feuille de papier. Ils sont comme des ordinateurs portables qui ne sont pas faits pour suivre l'utilisateur, mais qui sont à disposition partout et peuvent être utilisés par n'importe qui comme des « ordinateurs brouillons ». Enfin, les *boards* [Elrod *et al.*, 1992] font la taille d'un tableau blanc et offrent différentes utilisations notamment dans le cadre de réunions, présentations ou collaborations à distance.

Comme autre exemple nous pouvons citer le consortium *Things That Think* (http://ttt.media.mit.edu) du laboratoire MIT Media Lab. Depuis 1995, ce consortium vise à inventer le futur d'objets et environnements numériquement augmentés, à embarquer l'informatique dans des objets de tous les jours tels que des vêtements, bijoux et tables.

2.2.1.1 1995 – L'informatique calme

Pour mieux véhiculer son idée, Weiser introduit en 1995 le terme de technologie ou informatique *calme*. Il la définit comme une technologie qui engage à la fois le centre et la périphérie de notre attention, et qui se déplace même de l'un à l'autre [Brown et Duguid, 1996]. Lorsque nous conduisons une voiture, notre centre d'attention est porté sur la route mais pas sur le bruit du moteur qui est en périphérie. Si ce bruit devient inhabituel, il passera instantanément de la périphérie au centre de notre attention. De la même façon, la technologie calme va se déplacer facilement de la périphérie au centre. Placer des choses en périphérie nous permet d'être à l'écoute de plus de choses que ce que nous pourrions supporter si tout était au centre. Par exemple, une vidéo-conférence nous permet de suivre la conversation plus facilement qu'une audio-conférence car les expressions faciales et les gestes nous fournissent des informations supplémentaires sans effort supplémentaire, sans provoquer une surcharge d'informations.

Comme exemple d'applications de technologie calme, nous pouvons citer la « corde balançante » *(Dangling String)*, aussi appelé « fil de phase » *(Live Wire)*, créée par l'artiste Natalie Jeremijenko et installée à Xerox PARC[1]. Il s'agit d'un morceau de ficelle fixé à un moteur pas-à-pas et contrôlé via une connexion au réseau local. L'activité réseau se traduit par des mouvements saccadés du fil, lequel rapporte une indication périphériquement apparente du trafic. Ceci permet de percevoir sans effort explicite une information autrement inconnue – l'intensité du trafic réseau.

Donald A. Norman, dans son livre publié en 1999 et intitulé *The Invisible Computer* [Norman, 1999], adopte le même point de vue que Weiser. L'auteur pense que l'ordinateur de son époque est trop intrusif, trop difficile à utiliser, et inadapté à l'être humain. Il le compare même aux premiers phonographes, tellement malcommodes que l'entreprise de Thomas Edison a fini par faire faillite, alors que la technologie sous-jacente constitue une des grandes inventions de son siècle. Norman propose de créer des *information appliances*, des instruments (ou logiciels) pour l'information analogues aux appareils électroménagers. Chacun de ces outils serait aussi parfaitement adapté à une tâche particulière que l'est un aspirateur ou un lave-vaisselle. En addition, ces outils seraient capables de se communiquer de l'information facilement. Leur utilisation serait si naturelle pour les personnes, que l'ordinateur en deviendrait *invisible*.

[1] http://www.ubiq.com/hypertext/weiser/calmtech/calmtech.htm

2.2.2 1999 – Intelligence ambiante

L'*intelligence ambiante* ou *AmI* est le prolongement de l'informatique ubiquitaire (section 2.2.1) en y intégrant d'autres domaines, principalement celui des interfaces utilisateur intelligentes qui permet aux usagers de contrôler et interagir avec des objets de l'ubicomp de manière intuitive. La vision de l'AmI est donc toujours celle de l'ubicomp (des ordinateurs qui imprègnent l'environnement tout en étant transparents à l'utilisateur), mais en y ajoutant la notion d'intelligence, c'est-à-dire la faculté d'analyse du contexte et l'adaptation dynamique aux situations.

Les premières recherches en AmI ont eu lieu à Philips dès 1998. Ces recherches ont commencé par une série d'ateliers internes pour déterminer comment l'industrie de l'électronique grand public pourrait passer d'un monde d'appareils fragmentés à un monde en 2020 où des dispositifs conviviaux sont le support d'information, communications et divertissements ubiquitaires. Ces développements ont gagné en maturité et en 1999 Philips s'est joint à l'alliance *Oxygen*, un consortium international de partenaires industriels dans le contexte du projet *Oxygen* du MIT [Dertouzos, 1999] [2] dont le but était le développement de technologies pour l'ordinateur du 21ème siècle. En 2002, Philips a ouvert HomeLab[3] [Weber *et al.*, 2005], une infrastructure pour étudier la faisabilité et l'utilisabilité de produits issus de la recherche en AmI. HomeLab est un appartement équipé et surveillé, une photo du salon de cet appartement-laboratoire est montrée figure 2.4.

FIG. 2.4 – Le salon de HomeLab, un appartement dédié à l'étude de systèmes d'AmI à Philips.

En parallèle des recherches menées chez Philips, une série de réunions de l'ISTAG *(Information Societies Technology Advisory Group)* ont eu lieu afin d'élaborer la notion d'intelligence ambiante et de donner une vision guidant la direction globale du programme européen de l'IST *(Information Societies Technology)* dédié aux technologies de l'information. Le résultat de ces réunions est

[2] http://www.oxygen.lcs.mit.edu/
[3] http://www.research.philips.com/technologies/projects/homelab/index.html

un document [Ducatel *et al.*, 2001] publié en 2001 développant quatre scénarios illustrant l'intelligence ambiante pour 2010. Leurs objectifs étaient d'une part d'alimenter sur un long terme la recherche et d'autre part, d'évaluer la position scientifique de l'Europe à propos de ce domaine. Le premier scénario « *Maria, road warrior* », raconte la facilité de voyager grâce à l'intelligence ambiante. Celle-ci permet d'éviter les formalités aux frontières, dialoguant toute seule avec la douane, d'utiliser une voiture louée sans requérir de clés et comportant un système de guidage automatique. De plus, la chambre d'hôtel est personnalisée (température, musique, luminance). Un autre scénario, « *Carmen : traffic, sustainability and commerce* », présente des systèmes d'intelligence ambiante capables d'organiser du co-voiturage, de faire des courses dans un supermarché virtuel, tout en contrôlant ce qui est déjà dans sa cuisine. L'idée récurrente de l'ISTAG est que l'intelligence ambiante devra être attentive aux caractéristiques spécifiques de chacun, s'adapter aux besoins des utilisateurs, être capable de répondre intelligemment à des indications parlées ou gestuelles, et même d'engager un dialogue. Elle devra être non intrusive et le plus souvent invisible. Enfin, elle ne devra pas impliquer une longue période d'apprentissage pour l'usager et devra pouvoir être utilisée par les gens ordinaires. [Remagnino et Foresti, 2005] pensent également que pour rendre ces services adaptés à l'utilisateur, l'informatique ambiante devra pouvoir automatiquement interpréter leurs intentions.

L'intelligence ambiante commence par l'étude de la vie courante et l'exploration des manières acceptables dont la technologie peut être utilisée pour améliorer le quotidien des utilisateurs [Ducatel *et al.*, 2001, Abowd *et al.*, 2002]. La première chose proposée par [Abowd *et al.*, 2002] est de se diriger vers une interaction plus naturelle pour les humains que le clavier et la souris, une interaction incluant la parole et les gestes. Ensuite, les entrées utilisateur pourraient devenir de plus en plus implicites, la machine devenant capable d'inférer une signification à partir des signaux capteurs bas niveau. En addition, selon [Abowd *et al.*, 2002], le monde devrait être augmenté en fournissant de nombreux appareils hétérogènes offrant différentes formes d'interactions. La présentation d'informations à l'utilisateur pourrait également sortir du seul écran de l'ordinateur de bureau et utiliser toutes les surfaces d'affichage possibles. Certaines informations pourraient être présentées d'une manière non intrusive, dans la périphérie de l'attention de l'utilisateur. Enfin, les appareils en réseau devraient être orchestrés pour fournir une expérience holistique à l'utilisateur.

Au cours de l'année 2003, l'ISTAG a réexaminé la vision de l'AmI afin de vérifier sa validité et déterminer ce qui pourrait être fait pour la réaliser sur un moyen terme. Ceci a donné lieu à un autre rapport [Ducatel *et al.*, 2003]. L'ISTAG a estimé qu'il était nécessaire d'adopter une vision globaliste de l'AmI, en prenant en considération non seulement l'aspect technologique, mais l'ensemble de la chaîne d'innovation, depuis la recherche scientifique jusqu'à l'utilisateur final, en passant par les différentes facettes de l'environnement académique, industriel et administratif qui facilitent ou entravent la réalisation de la vision de l'AmI.

Plusieurs raisons justifient l'adoption d'une telle vue globale. D'abord, la complexité technique de systèmes modernes basés sur les technologies de l'information et de la communication exige que tous les acteurs de la chaîne de l'innovation intègrent leurs efforts. Ensuite, la co-évolution rapide de la technologie et du marché requiert la même chose. Enfin, la concentration et la cohérence nécessaires à la réalisation à la fois de développements technologiques importants

et d'un impact significatif sur le marché nécessitent l'engagement de chercheurs à la fois universitaires et industriels. Ceci est conditionné par la stratégie d'entreprise, qui est elle-même conditionnée par la stratégie d'investissements.

Ainsi, un certain nombre de collaborations entre des laboratoires de recherche et l'industrie ont émergé. L'on peut notamment citer le consortium AIR&D *(Ambient Intelligence Research and Development)* entre l'INRIA, Philips et Thompson démarré en 2001. Le premier projet né de ce consortium est le projet européen *Ozone* dont le but était de rendre l'interaction des utilisateurs avec les appareils plus conviviale, permettant l'émergence de services nouveaux et améliorés. Les trois orientations de ce projet étaient la reconnaissance vocale, l'architecture logicielle pour l'AmI et l'architecture matérielle. Un autre exemple est le projet *Ambience*, démarré en 2001, coordonné par Philips et incluant de nombreux partenaires industriels et universitaires. La *Fraunhofer-Gesellschaft* a démarré plusieurs activités dans divers domaines, y compris le multimédia, la conception de microsystèmes et les espaces augmentés. Le MIT a lancé un groupe de recherche sur l'AmI dans leur laboratoire Media Lab[4]. En 2004, le premier colloque européen sur l'intelligence ambiante (EUSAI) a eu lieu et de nombreuses autres conférences ont eu lieu, traitant de sujets particuliers en AmI.

2.2.2.1 Exemples de systèmes d'intelligence ambiante

Produits issus du HomeLab de Philips Le premier exemple de résultat de la recherche en AmI chez Philips est *Interactive Mirror* (basé sur *Mirror Display*), un miroir incorporant un écran LCD permettant d'afficher des informations sur le miroir de la salle de bain pendant que l'utilisateur se prépare le matin (par exemple la météo ou des dessins animés pour les enfants). Une photo de ce système est montrée figure 2.5. L'interactivité est ajoutée par la détection d'entrées utilisateur (interaction avec le cadre ou à proximité du miroir).

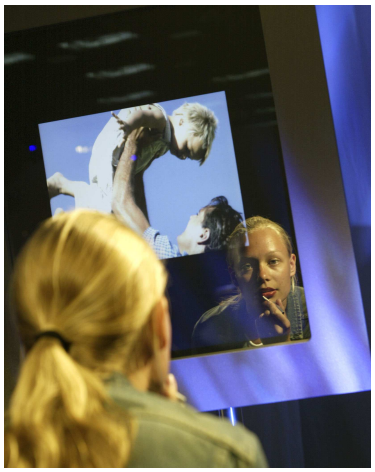

FIG. 2.5 – Le *Mirror Display* de Philips, un miroir qui est également un écran LCD.

D'autres exemples sont les *Interactive Toys*, des jouets mêlant des capacités d'interaction, et le *iCat*, un robot qui peut assister l'utilisateur par exemple

[4] http://ambient.media.mit.edu/

dans la cuisine. En combinant des informations de son agenda, son poids et ses activités, il peut donner des conseils sur ce que l'utilisateur peut ou ne devrait pas manger ou faire (tout en prenant en compte les préférences personnelles).

Un autre projet de recherche en AmI chez Philips est le projet *am*BX, dont le but était de rendre les jeux vidéo plus réalistes. Un simple langage associait à un jeu des effets dans l'environnement réel (par exemple faire clignoter les lampes de la maison lorsque le personnage du jeu est surpris par un ennemi). Un produit issu de la même idée et qui se voit couramment dans les magasins est *Ambilight*, une télévision illuminant le mur derrière elle avec la couleur dominante de l'image afin d'augmenter l'effet d'immersion dans le programme regardé.

Ces applications ont notamment été incluses dans le projet européen *Amigo* dont Philips était l'un des partenaires et dont il sera question dans la section 2.2.6.2.

Produits issus du projet *Oxygen* du MIT La vision du projet *Oxygen* était de rendre l'informatique centrée utilisateur, disponible gratuitement partout dans le monde, comme les piles et les prises de courant, ou l'oxygène. Ils voulaient faire entrer l'informatique dans le monde des humains, lui faire gérer nos objectifs et besoins, et nous aider à faire plus en faisant moins. Nous n'aurions alors plus besoin de transporter nos propres appareils avec nous, mais au lieu de cela, l'informatique nous serait fournie par des appareils génériques, soit portatifs, soit intégrés dans l'environnement. Lorsque nous interagirions avec ces appareils « anonymes », ils adopteraient nos personnalités informatiques. Ils respecteraient bien sûr notre vie privée et sécurité, et nous n'aurions pas besoin d'apprentissage pour pouvoir nous en servir. Nous communiquerions avec eux de manière naturelle, utilisant la parole et les gestes.

En suivant cette vision, le projet *Oxygen* a créé deux sortes d'appareils : les *E21s*, les appareils embarqués dans l'environnement, et les *H21s*, les appareils portatifs.

Le *H21* est montré figure 2.6. Il s'agit d'un PDA augmenté avec une caméra, un accéléromètre, un circuit FPGA, un logiciel de traitement du signal, un haut-parleur, un microphone et un détecteur infrarouge. Le *H21* est anonyme par défaut, mais lorsqu'un utilisateur le prend en main, il se personnalise avec ses préférences.

FIG. 2.6 – L'appareil « générique » mobile *H21*, prototype du projet *Oxygen*.

Les *E21* sont des appareils dispersés dans l'environnement pour fournir une zone locale de calcul et de communication à un espace intelligent. Les *E21s* sont

connectés aux capteurs, actionneurs et appareils voisins. Ils sont convenablement encapsulées dans des objets physiques. Ils communiquent les uns avec les autres et avec les *H21s* à proximité par le biais de réseaux dynamiquement configurés (*N21s*, également développés par *Oxygen*). Par exemple, un *E21* pourrait contrôler une rangée de microphones utilisée par les ressources perceptuelles d'*Oxygen* afin d'améliorer la communication avec les haut-parleurs en filtrant le bruit de fond.

De plus, le projet *Oxygen* a développé un système de localisation d'intérieur, *Cricket*. Ils ont ensuite équipé une pièce témoin avec ce système *(the Intelligent Room)* en ajoutant des rangées de microphones et de caméras pour écouter les usagers et observer leur activité. Une application est *MeetingView* qui enregistre de façon inoffensive la progression d'une réunion dans un *Intelligent Room*, puis qui montre le contenu d'une manière qui encapsule le format de la réunion et fournit des outils facilitant l'analyse.

Produits issus de l'INRIA En 2002, l'INRIA et la cité des sciences ont mené conjointement un travail consistant à développer un système de navigation spatiale destiné aux visiteurs, sous la forme d'un PDA, appelé « le navigateur ». Il s'agit dans un premier temps d'un dispositif d'accompagnement et de suivi personnalisé des visiteurs. Il utilise un identificateur, le ticket d'entrée pourvu d'un code barre, qui permet d'enregistrer le parcours dans l'exposition. Il permettra ensuite d'accéder à des informations complémentaires sur sa visite, sur place ou par internet. À terme, le ticket prendra la forme d'un PDA, enregistrant des paramètres personnels pour proposer une offre adaptée. Il permettra d'organiser la visite en constituant un parcours, en gérant par exemple l'affluence ou les horaires des projections.

D'autres exemples de systèmes d'AmI ou d'ubicomp sont présentés dans la section 2.2.6.

2.2.3 2005 – Constat sur les accomplissements de l'informatique ubiquitaire

L'informatique ubiquitaire est un domaine mené depuis une quinzaine d'années par une vision très prononcée. En 2005, le domaine avait atteint une certaine maturité permettant aux chercheurs de regarder en arrière et porter un jugement critique sur ses accomplissements et ses défaillances. C'est ce qu'ont notamment fait Geneviève Bell et Paul Dourish dans leur article [Bell et Dourish, 2007] écrit en 2005 et paru en 2007 dans le journal *Personal and Ubiquitous Computing*. Cet article, intitulé *Yesterday's tomorrows : notes on ubiquitous computing's dominant vision*[5], est une analyse de l'état de l'art en informatique ubiquitaire et un constat du fait que la plupart des publications du domaine parlent de l'informatique ubiquitaire comme d'un futur proche, exactement comme le faisait Weiser en 1989. C'est donc un domaine qui est toujours tout proche *(just around the corner)* depuis 15 ans, mais qui est toujours repoussé à demain, qu'on n'atteint finalement jamais.

Weiser parlait de l'ordinateur du 21ème siècle, or nous sommes aujourd'hui au 21ème siècle mais nous pensons n'avoir toujours pas atteint sa vision. L'article

[5] « Les demains d'hier, notes sur la vision dominante de l'informatique ubiquitaire »

affirme que cette vision a, en réalité, été atteinte, mais à des détails près. Ces différences sont le fruit de nos attentes technologiques qui sont illusoires.

Pour illustrer leur argument, les auteurs décrivent deux exemples contemporains : Singapour et la Corée. Par des exemples d'utilisation des nouvelles technologies dans ces deux pays, les auteurs prouvent que l'on peut les désigner comme *environnements ubiquitaires*. En effet, Singapour porte le surnom de *intelligent island* (« l'île intelligente ») et la Corée est porteuse d'un plan technologique visant à transformer le pays en une société ubiquitaire en 2010, communément appelé *U-Korea* dans la presse. D'après les auteurs, la vision de Weiser est donc bel et bien une réalité aujourd'hui, si l'on accepte de l'adapter à la réalité technologique contemporaine.

Dans la vision de Weiser, les infrastructures devaient être continues, sans plis et sans heurts[6], ce qui n'est pas le cas aujourd'hui et ceci est la raison pour laquelle nous repoussons toujours l'informatique ubiquitaire à demain. Mais, d'après les auteurs, les infrastructures sont par nature un fouillis[7], elles doivent en permanence être maintenues et renégociées. *Le futur est déjà ici, il n'est juste pas très régulièrement distribué* [Gibson, 1999]. La conclusion des auteurs est que l'informatique ubiquitaire est une réalité dans la société d'aujourd'hui (ou dans certaines sociétés du moins), mais qu'elle n'est pas aussi propre et ordonnée que ce que nous attendions. Les appareils la composant ne sont pas discrets et invisibles comme Weiser les avait imaginés, mais au contraire très présents et visibles (l'on peut notamment penser aux nouveaux téléphones tels que l'*iPhone* qui ont un grand succès commercial, ou encore aux objets communicants grand public dont quelques exemples seront présentés dans la section 4.1.1.1). La bonne approche est peut-être de laisser les gens inventer leurs usages de l'informatique ubiquitaire, plutôt que d'essayer de leur en imposer sans qu'ils ne s'en aperçoivent. Le domaine de l'informatique ubiquitaire a donc besoin d'être remis à jour.

Cette idée est également développée par [Rogers, 2006]. L'auteur résume les efforts fournis depuis une quinzaine d'années pour réaliser des applications d'« informatique calme » (notion décrite dans la section 2.2.1), mais constate qu'ils n'égalent toujours pas la vision originale. La raison en est simplement qu'atteindre cet objectif implique résoudre des problèmes d'intelligence artificielle trop difficiles et que nous ne savons pas encore résoudre. De plus, le comportement des gens dans la « vraie vie », leurs humeurs et intentions, sont trop complexes pour être correctement intégrés dans un système informatique. L'auteur évoque également le fait que l'informatique calme, qui s'efface et est transparente pour les utilisateurs, n'est peut-être pas souhaitée par ces mêmes utilisateurs. Ceux-ci ne désirent pas dépendre des ordinateurs dans leurs tâches quotidiennes, mais tiennent, pour la plupart, à être actifs et garder le contrôle. L'auteur propose alors d'élargir la portée du domaine de l'informatique ubiquitaire au regard de l'évolution des pratiques constatée ces dernières années et qui est que les technologies *engagent* les utilisateurs. Les applications populaires ne sont pas celles qui sont transparentes et invisibles, mais celles qui engagent la créativité et l'imagination des utilisateurs. C'est vers cette direction, d'après l'auteur, que doit désormais se tourner l'informatique ubiquitaire.

Des problèmes dans l'approche de l'informatique ubiquitaire ont également

[6] Le terme anglais employé est *seamless*.
[7] Le terme anglais employé est *messy*.

été remarqués par John Barton et Jeff Pierce dans leur papier de positionnement [Barton et Pierce, 2006] présenté en 2006. Ce papier porte une critique sur les scénarios utilisés pour évaluer les systèmes ubiquitaires, les caractérisant de peu réalistes. Selon les auteurs, ces scénarios sont trop futuristes, intéressants du point de vue technique, mais souvent inutiles dans la pratique car ils simplifient trop la nature mondaine de notre vie de tous les jours ; aussi proposent-ils d'évaluer les scénarios-mêmes en quantifiant la « magie » qu'ils contiennent. En effet, les scénarios inappropriés résultent en des systèmes non réalistes que personne ne va adopter, en des évaluations biaisées et en le risque de rater le potentiel réel d'une technologie.

C'est pour ces raisons que Rui José parle d'innovation ouverte remplaçant les scénarios dans son exposé plénier à la conférence UCAmI en 2008 [José, 2008]. Selon lui, il serait plus fructifiant de concevoir des artefacts et des services de façon à ce que ce soient les utilisateurs eux-mêmes qui leur donnent du sens et une place dans leurs vies de différentes manières. L'auteur plaide en faveur d'un dialogue continu entre la recherche en informatique ubiquitaire et la société.

Un exemple de travaux sur l'idée que c'est leur usage qui donne de l'intelligence aux systèmes est donné par [Taylor *et al.*, 2007], article écrit en 2005 et publié en 2007. Cet article présente quatre applications implémentant la notion selon laquelle la meilleure approche n'est pas d'essayer de créer des environnements intelligents mais plutôt de créer des applications dont la réalisation est simple, qui ne sont pas ce qu'on appelle intelligentes par leur conception, mais dont l'usage par les personnes les rend intelligentes.

En effet, selon les auteurs, nos chances de réussir à construire un véritable système intelligent pour la maison sont minces. Un tel système sera si complexe que les bénéfices qu'il apporte à un utilisateur non expert devront être énormes pour justifier son acceptation par l'usager. Pour cette raison, de nombreuses recherches sont menées pour rendre intelligent l'habitat d'une personne à autonomie réduite. Dans ce cas, les contraintes sont bien définies et les bénéfices sont immédiats.

De même que [Rogers, 2006] et [Bell et Dourish, 2007], [Taylor *et al.*, 2007] prônent les systèmes dont l'intelligence est située dans l'usage que les gens ont décidé d'en faire (« système qui *engage* l'utilisateur » [Rogers, 2006]) et non dans leur conception. [Edwards et Grinter, 2001] donnent également des exemples de technologies ayant été adoptées par les utilisateurs de manière surprenante et imprévisible, l'exemple le plus frappant étant simplement le téléphone.

Les quatre exemples d'applications présentées par [Taylor *et al.*, 2007] sont des applications pour la maison qui visent à augmenter les pratiques déjà existantes avec des objets ordinaires[8]. La première application est un aimant pour réfrigérateur qui permet aux divers éléments que la famille place sur la porte du frigo d'être annotés. Cette annotation peut simplement être l'identité du créateur de l'élément (l'aimant affiche alors la photo de cette personne) ou bien un court enregistrement vocal. Ces aimants peuvent également briller soit pendant 24 heures après avoir été déplacés (afin d'attirer l'attention sur un changement), soit ils peuvent briller un jour précis de la semaine (afin de rappeler un événement).

[8] D'autres exemples peuvent être trouvés à l'adresse http://research.microsoft.com/en-us/groups/sds/default.aspx, dans le cadre du groupe de recherche sur les systèmes socio-digitaux de Microsoft Research dont les auteurs font partie.

La deuxième application présentée est un post-it numérique, appelé *HomeNote*, placé encore une fois sur le frigo (les auteurs soutiennent que la cuisine est un lieu central de vie dans une maison). Cette interface permet l'écriture de notes manuscrites et la réception de messages SMS.

La troisième application, appelée *the Whereabouts Clock*, est une interface divisée en plusieurs catégories de lieux (par exemple « maison », « travail », « école » et « ailleurs ») et affichant en permanence une icône pour chaque membre de la famille dans le lieu où il se trouve (la localisation est grossière et basée sur l'antenne la plus proche du téléphone portable de l'usager).

Enfin, la quatrième application est appelée *the picture bowl* et se présente comme un bol, qui se trouverait typiquement à l'entrée de la maison et dans lequel la famille peut déposer toutes sortes d'objets, y inclus ses téléphones portables et appareils photos. Leurs contenus sont alors copiés et affichés sur les parois du bol. Les contenus les plus récents sont en haut du bol et descendent au fur et à mesure que d'autres contenus sont ajoutés. Quelques fonctionalités d'interaction simples avec les vignettes sont prévues.

Toutes ces applications sont techniquement simples à réaliser. Si elles paraissent intelligentes, c'est parce qu'elles augmentent les usages déjà existants (mettre des notes sur le frigo ou se débarrasser de son bazar dans un bol dans l'entrée). En outre, l'application *HomeNote* a été particulièrement bien adoptée par les testeurs car son usage a eu un réel apport dans les interactions de la famille : ils ont trouvé un usage à valeur ajoutée à un objet techniquement simple de réalisation. Cet appareil leur a permis d'envoyer des messages qui ne sont pas destinés à quelqu'un en particulier, mais à toute la famille, et qui sont sémantiquement rattachés à un lieu. Ces messages n'étaient pas nécessairement de nature « utile », mais pouvaient être simplement affectifs, permettant d'être présent dans la maison, de se faire remarquer de manière non intrusive par la famille. Cet appareil véhicule donc des valeurs humaines, ce qui fait son succès (ce qui soutient l'argument de [José, 2008]).

Dans l'article [Pascoe *et al.*, 2007], on retrouve cette idée d'applications simples et réellement utilisées. Cet article présente une étude menée sur un groupe d'utilisateurs afin de déterminer leur façon de prendre en compte le contexte dans leurs usages d'appareils électroniques (leurs téléphones mobiles ou PDAs). Une des choses ressorties de l'enquête est que l'un des comportements les plus observés chez les sujets était de créer des notes ou des rappels sur leurs appareils mobiles, ces notes étant contextuelles. Mais cette fonctionnalité est une des moins prises en charge par l'informatique. Le processus d'inclure une information de contexte dans une note était toujours manuel. Pourtant, cette idée a été étudiée [Brown, 1996] et pourrait volontiers être employée si elle était disponible. Cette approche est celle prônée par [Taylor *et al.*, 2007], c'est-à-dire d'étudier le comportement des gens et de proposer des applications informatiques basées sur leurs habitudes actuelles.

Tout comme [Rogers, 2006] et [Taylor *et al.*, 2007], [Leahu *et al.*, 2008] pensent qu'il n'est pas nécessaire de résoudre un problème difficile et complet d'intelligence pour créer des applications utiles. Les auteurs font un rapprochement entre l'intelligence artificielle (IA) et l'informatique ubiquitaire (ubicomp). Selon eux, le problème central de l'ubicomp est identique à celui de l'IA, à savoir comment des systèmes informatiques peuvent donner un sens et réagir raisonnablement à un environnement complexe et dynamique chargé de significations humaines. En particulier, l'ubicomp est actuellement confrontée à une série de

défis dans le passage à l'échelle de prototypes qui travaillent dans des environnements restreints à des solutions qui fonctionnent de manière fiable et robuste dans des environnements où toute la complexité apportée par les humains est prise en compte.

L'IA n'a pas encore trouvé de solution générale à ce problème, mais plusieurs approches alternatives ont émergé, notamment l'IA *interactionniste*, qui fournit des approches concrètes, techniquement réalisables, pour concevoir des interactions intelligentes et en temps réel avec un environnement constamment en mutation. Les auteurs présentent alors six stratégies développées par l'IA interactionniste afin de créer des systèmes intelligents tout en évitant les problèmes bloquants en IA classique, et comment ces techniques pourraient être transposées en ubicomp. Ces techniques sont notamment justifiées par l'article [Bell et Dourish, 2007] décrit ci-dessus, qui soutient que l'hypothèse de l'IA classique que le monde est ordonné et organisé est fausse et que le monde réel est plutôt en désordre et hétérogène. Ces stratégies suggèrent de déplacer l'objectif de la conception de systèmes ubiquitaires intégrés et complets à la construction de nombreux petits dispositifs ad-hoc qui donnent l'avantage aux régularités de l'environnement et des comportements et perceptions des gens, et qui participent de manière utile et engageante dans le monde humain.

Pour clore cette section, nous pouvons citer le livre d'Adam Greenfield paru en 2006 et intitulé *Everyware : The Dawning Age of Ubiquitous Computing*[9] [Greenfield, 2006]. Greenfield décrit le paradigme d'interaction de l'informatique ubiquitaire comme le « traitement de l'information se dissolvant dans le comportement ». Dans une série de méditations brèves et réfléchies, l'auteur explique comment *everyware* (le nom qu'il donne à tous ces aspects de l'informatique ubiquitaire qui sont déjà présents dans notre quotidien) est déjà en train de modifier notre vie, en transformant notre compréhension des villes où nous vivons, des communautés auxquelles nous appartenons – et l'image que nous avons de nous-mêmes.

2.2.4 2008 – La vision du futur (« ubicomp 2.0 »)

À partir de ce constat (section 2.2.3), quelle est finalement la direction à suivre pour les recherches futures en informatique ubiquitaire ?

[José, 2008] nous incite à nous focaliser sur l'expérience utilisateur *(user experience)*, à comprendre les individus et leurs expériences avec les technologies, de transférer l'accent du « comment » au « pourquoi » : quelles technologies sont souhaitables d'un point de vue socioculturel et pourquoi une personne accepterait ou rejetterait-elle une certaine technologie. Il suggère également de tenir compte des valeurs humaines telles que la vie privée, la sécurité, la connexité, la camaraderie[10], la créativité ou encore la propriété. L'auteur va plus loin en préconisant d'intégrer totalement la prise en compte de ces valeurs dans la conception des systèmes. Enfin, l'auteur introduit un nouveau terme pour désigner ces changements de direction dans l'informatique ubiquitaire : *ubicomp 2.0*.

Des projections dans le futur sont faites notamment par Microsoft dans un rapport [Harper *et al.*, 2008] publié en 2008 sur l'état de l'interaction homme-

[9] « L'âge de la naissance de l'informatique ubiquitaire »

[10] Ce sont justement les effets observés sur les utilisateurs de l'application *HomeNote* [Taylor *et al.*, 2007] décrite dans la section 2.2.3 : les membres de la famille envoyaient des messages à la maison afin de rester connectés avec leurs proches et présents dans la maison.

machine en 2020. Ce rapport fait d'abord un état de l'art sur le rôle de l'ordinateur et des nouvelles technologies en général dans la vie de tous les jours en constatant que désormais, les ordinateurs peuvent être impliqués dans presque n'importe quel aspect de nos vies. De plus, ce lien de l'informatique à nos activités quotidiennes peut, dans le futur, affecter à son tour nos valeurs, objectifs et aspirations. Ensuite, ce rapport révèle cinq transformations de nos interactions avec les ordinateurs qui devraient avoir lieu alors que nous approchons 2020. La façon que nous avons de définir notre relation avec les ordinateurs subit un changement radical. La façon dont nous les utilisons et en dépendons est également en cours de transformation. En même temps, nous devenons de plus en plus connectés et nos actions, conversations et interactions sont de plus en plus gravées dans nos « paysages numériques »[11].

Les auteurs pensent qu'il y a une portée plus importante que jamais dans la résolution de problèmes difficiles et la création de nouvelles formes d'engagement et de créativité. Mais ces nouvelles perspectives soulèvent des questions et des préoccupations. Les ordinateurs étant de plus en plus intégrés dans la vie quotidienne, l'interaction indirecte va-t-elle être acceptée par la société ? L'informatique ubiquitaire vise à créer un écosystème informatique dans lequel les ordinateurs sont non seulement présents pour les utilisateurs, mais travaillent de plus en plus ensemble. Comment allons-nous maîtriser un écosystème d'une telle complexité, qui sera responsable de ses erreurs, peut-il y avoir des effets globaux indésirables ? L'augmentation de notre dépendance à la technologie va-t-elle rendre certaines de nos compétences naturelles obsolètes ? Dans quelle mesure la société va-t-elle accorder à des ordinateurs intelligents la confiance qu'elle a actuellement en des professionnels et experts qualifiés ?

Une question qui se pose déjà aujourd'hui : quel sera l'impact de grands réseaux sociaux sur nous, nos familles et amis, et la société en général ? Comment les technologies peuvent-elles être utilisées pour assembler et mobiliser de manière efficace des groupes de personnes afin de lutter contre les problèmes mondiaux ? Comment la société devrait-elle gérer le stockage et l'accès aux données privées des personnes de manière éthique et responsable ? De qui est-ce la responsabilité de veiller à ce que les systèmes de surveillance soient conçus en équilibre avec les droits des individus et ceux de la société ? Les scientifiques risquent-ils de devenir trop dépendants des outils informatiques dans leurs recherches ? Si tel est le cas, que cela signifie-t-il sur la nature de l'invention et de la découverte ?

Face à ces observations et questions, la conclusion de ce rapport sur la marche à suivre est la suivante. L'interaction homme-machine a besoin de se préoccuper moins de la production et du traitement des informations et davantage de la conception et de l'évaluation de systèmes qui permettent d'atteindre des valeurs humaines. Cela exige que les contraintes épistémologiques en interaction homme-machine permettent une conception sensible à la signification, raison d'être et caractère souhaitable des technologies. Ceci suggère l'ajout d'une cinquième étape de la conception et recherche classiques en interaction homme-machine : une étape d'analyse conceptuelle où l'on considère les valeurs humaines que l'on cherche à appuyer. Cela affecte l'ensemble du cycle de recherche et de conception, y compris la manière de comprendre l'utilisateur, de faire des études sur le

[11] Autrement dit, la trace que laissent nos activités digitales dans les historiques, les mémoires de nos ordinateurs ou de ceux des sites internet que nous visitons.

terrain et en laboratoire, de réfléchir sur les valeurs recherchées, de construire des prototypes et d'évaluer les modèles. Enfin, les chercheurs en interaction homme-machine ont besoin d'un éventail plus large de compétences et de savoir-faire afin de réaliser ces buts.

2.2.5 Définition de la notion de contexte

Au cours des sections précédentes, nous avons à plusieurs reprises évoqué la notion de contexte. Nous allons ici en donner une définition et une description plus complètes.

La définition générale du contexte que l'on trouve dans les dictionnaires est la suivante : « Ensemble que forment, par leur liaison naturelle, les différentes parties d'un texte ou d'un discours ». On peut également souligner les définitions suivantes[12] :

- « Le contexte d'un événement inclut les circonstances et conditions qui l'entourent » ;
- « En informatique, le contexte est l'ensemble des conditions sous lesquelles un dispositif est en train d'être utilisé, par exemple l'occupation actuelle de l'utilisateur » ;
- « En intelligence artificielle, le contexte est très fortement relié à ses propriétés en communication, linguistique et philosophie. La recherche scientifique est effectuée sur la façon dont ces aspects peuvent être modélisés dans des systèmes informatiques (par exemple basés sur la logique) pour l'utilisation dans le raisonnement automatique ».

Clairement définir la notion de contexte pour l'informatique ubiquitaire a intéressé différents chercheurs. En effet, il est difficile de trouver une définition claire et unique, valable dans tous les travaux impliquant cette notion. Par exemple, pour [Schilit et Theimer, 1994], le contexte est « la localisation et l'identité des personnes et objets à proximité ». Pour [Ryan *et al.*, 1998], il s'agit plutôt de « la localisation, l'identité et l'heure ». Dans un article plus poussé sur le sujet, [Dey *et al.*, 2001] donnent la définition suivante : « toute information pouvant être utilisée pour qualifier la situation des entités [...], typiquement la localisation, l'identité et l'état des personnes, groupes et objets informatiques et physiques ». La définition la plus vaste a été donnée par [Schilit *et al.*, 1994], l'un des premiers articles à aborder le sujet :

> « Context encompasses more than just the user's location, because other things of interest are also mobile and changing. Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situation ; *e.g.*, whether you are with your manager or with a co-worker. »

Paul Dourish aborde ce sujet dans [Dourish, 2004]. D'abord, pour l'auteur, *context-aware computing*[13] est l'art de créer des systèmes pouvant détecter des aspects de la configuration dans laquelle ils sont utilisés, et agir en fonction. Pour les sociologues, le fait que la plupart des systèmes interactifs n'aient pas cette capacité, est un manque [Suchman, 1987]. La prise en compte du contexte devrait rendre les systèmes plus réactifs aux diverses configurations sociales dans lesquelles ils sont utilisés.

[12] Source : Wikipedia (http://fr.wikipedia.org/wiki/Contexte)
[13] L'informatique sensible au contexte

Dans un de ses premiers articles sur l'informatique ubiquitaire, [Weiser, 1993] explique que l'idée de l'ubicomp est née de l'observation de la place qu'a l'ordinateur dans les activités de la vie quotidienne. En particulier, des études anthropologiques montrent que la vie professionnelle est régie par des situations partagées et des compétences technologiques non examinées. La notion d'« actions situées » de Suchman est une source de l'idée que les systèmes informatiques devraient répondre aux configurations de leurs usages. [Abowd *et al.*, 2002] citent également Suchman en disant que le comportement humain est en priorité improvisé, par opposition à un plan prédéfini a priori que la personne exécuterait. L'informatique ubiquitaire basée sur la notion d'actions situées fonctionnerait également selon l'idée de l'improvisation comme comportement, et non d'un script prédéfini que l'usager suivrait.

En 2006 est paru un article de John Canny [Canny, 2006] investiguant le passé, présent et futur du domaine de l'interaction homme-machine. Selon lui, les technologies sensibles au contexte sont le futur. L'auteur compare les nouvelles technologies qui lui sont contemporaines aux premières voitures : des « calèches sans chevaux » qui avaient encore des rênes, pour montrer leur inadéquation avec leurs nouveaux usages. Il donne l'exemple du téléphone portable qui a clairement été conçu pour l'interaction vocale et non textuelle. Pour l'auteur, le contexte a trois facettes : le contexte immédiat, le contexte de l'activité (incluant l'historique de l'utilisateur et de quelques autres personnes dans le cas d'une activité collaborative) et le contexte de la situation (permettant de savoir comment les autres acteurs se comportent généralement dans cette situation). L'auteur souligne également qu'une même technologie peut être un grand succès ou échec selon le contexte dans lequel elle est employée. Par exemple, la reconnaissance vocale pour les ordinateurs de bureau n'a pas marché car l'interaction textuelle est la plus pertinente et pratique dans ce cas. Par contre elle promet d'avoir un plus grand succès dans le domaine des téléphones portables et de l'informatique médicale. Comme nous le développerons également dans la section 2.2.4, le système doit s'adapter à son utilisation. Enfin, le contexte peut jouer un rôle pour améliorer les algorithmes perceptifs des appareils.

Dans notre cadre d'un environnement intelligent, *le contexte est l'ensemble des états des dispositifs et personnes dans l'environnement à un instant t*. Ces dispositifs et personnes évoluent de manière continue et, en partie, non prévisible.

Nous allons nous focaliser sur la définition de [Crowley *et al.*, 2002, Coutaz *et al.*, 2005, Crowley, 2006] qui s'inspire du théâtre comme source de concepts pour l'observation socialement avisée de l'activité humaine. [Crowley, 2006] soutient que l'activité humaine de tous les jours peut souvent être décrite, comme une pièce de théâtre, sous la forme d'un script. Ce script contient des définitions de rôles et une série de scènes composées elles-mêmes d'une série de situations. La définition d'un rôle contient un espace d'actions possibles, incluant des dialogues, mouvements et expressions émotionnelles. Ainsi, un grand nombre d'activités peut être décrit sous cette forme, notamment un cours à l'école, une réunion de travail, du shopping ou un dîner dans un restaurant. La grande différence entre une pièce de théâtre et la vie, concernant ces activités scénarisables, est qu'un script théâtral suit une séquence fixée de situations alors que la vie réelle est beaucoup moins contrainte et peut plutôt être vue comme un réseau ou graphe de situations, comprenant des boucles et des branches non déterministes. En cela Crowley est en cohérence avec le point

de vue de [Abowd *et al.*, 2002] et [Suchman, 1987] qui considèrent le comportement humain comme étant improvisé. Pour Crowley, l'improvisation réside dans la séquence des transitions empruntées dans le graphe des situations, mais les situations elles-mêmes (et les transitions possibles) peuvent être scénarisées. [Crowley *et al.*, 2002] proposent alors une hiérarchie de définitions de concepts pour l'observation de l'activité humaine, appelée *modèle de contexte* :

Entité Une personne ou un objet physique ou informatique. Une entité est définie par un ensemble de variables observables corrélées. *Par exemple, un vidéoprojecteur peut être modélisé comme une entité avec pour propriétés, la pièce dans laquelle il se trouve, l'ordinateur auquel il est branché, etc.*

Rôle Un filtre permettant de sélectionner, parmi toutes les entités perçues, celles qui sont pertinentes pour ce que l'on veut traiter. L'entité sélectionnée « joue » alors le rôle. *Par exemple, Bob peut jouer le rôle de conférencier s'il se tient à côté du tableau et parle.*

Relation Un prédicat sur les propriétés d'une ou plusieurs entités jouant des rôles. *Par exemple, « à l'intérieur de » est une relation entre deux entités dont une joue le rôle de « bureau » et qui dépend de leurs positions.*

Situation Un ensemble de rôles et de relations. *Par exemple, « donner une présentation » est une situation définie par les rôles « conférencier », « auditeur » et les relations « près du tableau » et « différent de ».*

Contexte Une composition de situations partageant toutes le même ensemble de rôles et de relations pertinents pour la tâche donnée. *Par exemple, le contexte « conférence » inclut la situation « présentation ».*

Définir le modèle de contexte (ou *modèle de situations*) pour une application donnée revient à définir les entités, rôles, relations, situations et arcs entre situations pertinents pour l'application. [Crowley *et al.*, 2002] pensent que le contexte est un concept clé pour l'informatique ambiante *(context is key)*. Chaque *situation* est un état particulier de l'environnement, pertinent pour l'application ; Elle est définie par une configuration particulière des entités, rôles et relations. Les rôles et les relations sont choisis pour leur pertinence par rapport à la tâche. De même, seules les entités qui peuvent éventuellement jouer des rôles et être pertinentes pour la tâche sont prises en compte. Une situation représente une affectation d'entités à des rôles, complétée par un ensemble de relations entre les entités. Une situation peut être considérée comme « l'état » de l'utilisateur à l'égard de sa tâche. Si les relations entre entités changent, ou si la liaison entre les entités et les rôles change, alors la situation dans le contexte a changé. Pour détecter les changements de situation, une fédération de processus d'observation est nécessaire. Afin de fournir des services, des règles associant des actions à des situations sont définies. Ces actions sont déclenchées lorsque la situation dans laquelle se trouve le système change. Les services correspondants sont alors fournis aux utilisateurs.

Prenons par exemple le contexte d'un séminaire de travail. Les entités pertinentes sont les personnes. Les rôles possibles sont *conférencier* et *auditeur*. La seule relation utile est *différentDe*. Les situations sont **Vide**, **Public** et **Présentation**. Enfin, les actions possibles sont allumerLumière et allumerProjecteur[14]. Le modèle de contexte « Séminaire » est alors représenté par le graphe de la

[14] Pour cet exemple, nous définissons un modèle minimal.

figue 2.7. Dans la situation **Présentation** (qui correspond à la phase où le conférencier fait son exposé), une personne doit jouer le rôle *conférencier* et les autres jouent le rôle *auditeur*. La relation qui doit être satisfaite est que l'entité qui joue le rôle *conférencier* est *différentDe* les entités jouant le rôle *auditeur*. La situation **Public** correspond à la phase où l'audience est présente dans la salle mais la présentation n'a pas encore commencé (il faut donc au moins une entité jouant le rôle *auditeur*). Enfin, la situation **Vide** représente la situation initiale et finale où personne n'est (encore/plus) présent dans la salle. Lorsque la situation **Public** est enclenchée, l'action allumerLumière est exécutée ; de même, lorsque le conférencier est en place, la situation **Présentation** devient active et l'action allumerProjecteur est exécutée. Les affectations d'entités aux rôles ne sont pas bijectives. En effet, une ou plusieurs entités peuvent jouer le même rôle en même temps. De même, une entité peut jouer plusieurs rôles à la fois. Ces affectations peuvent changer dynamiquement. Ainsi, le comportement des humains dans le contexte peut faire passer l'environnement d'une situation à l'autre en respectant les flèches entre situations. L'enchaînement des situations à l'intérieur du contexte est appelé *script*.

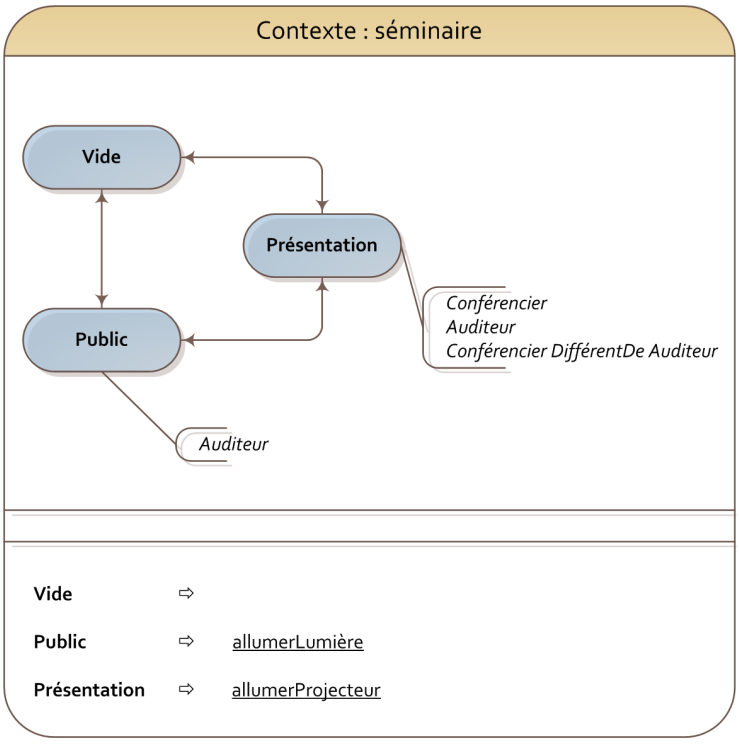

FIG. 2.7 – Un exemple de modèle de contexte : séminaire. **Vide**, **Public** et **Présentation** sont les situations, *conférencier* et *auditeur* – les rôles, *différentDe* – la relation, et allumerLumière et allumerProjecteur sont les actions.

Des exemples de systèmes sensibles au contexte sont présentés ci-dessous, dans la section 2.2.6.1.

2.2.6 Exemples de systèmes ubiquitaires

2.2.6.1 Systèmes sensibles au contexte

De nombreux travaux en ubicomp et AmI étudient des systèmes prenant en compte le contexte. Avant d'en présenter quelques exemples, il est incontournable de citer le *Context Toolkit* de [Dey et Abowd, 2000], une plate-forme facilitant le développement d'applications sensibles au contexte. Le *Context Toolkit* est constitué de *widgets*[15] de contexte et d'une architecture distribuée pour les accueillir. Ces *widgets* sont des composants logiciels offrant aux applications un accès à des informations sur le contexte tout en masquant les aspects matériels liés à la capture de ces informations.

[Yan et Selker, 2000] du Media-Lab ont proposé un assistant virtuel placé à la porte du bureau de l'utilisateur pour gérer les visiteurs en fonction du contexte de l'utilisateur à l'intérieur du bureau. L'assistant interagit avec les visiteurs par la voie de la conversation naturelle en face-à-face. Il gère l'agenda de son propriétaire (pour savoir si le visiteur a un rendez-vous et pour éventuellement fixer automatiquement un rendez-vous avec le visiteur) et utilise l'information de la présence éventuelle d'autres personnes dans le bureau.

[Chen *et al.*, 2004a] ont développé une architecture appelée le *context broker*[16] ou *CoBrA*, dont un schéma est montré figure 2.8. Dans la vision de Chen, l'environnement intelligent est muni de plusieurs *context brokers*, chacun étant responsable d'une partie de l'environnement (une pièce particulière par exemple). Le rôle d'un *context broker* est de maintenir un modèle du contexte dont il est responsable et de partager ce modèle avec d'autres agents ou services. Le *context broker* est un agent central qui reçoit des informations de différentes sources (des capteurs, d'autres agents, des appareils, des serveurs d'information, etc.) et doit fusionner ces informations afin de fournir un modèle toujours cohérent. De plus, le *context broker* est sensible à la vie privée des utilisateurs. Il respecte les degrés de confidentialité des informations qu'il manipule et ne divulgue à d'autres agents que les informations autorisées.

Le *context broker* utilise des ontologies (écrites en langage OWL) pour décrire les informations contextuelles et partager ses connaissances (cette partie est décrite en détail dans [Chen *et al.*, 2004b]). De plus, il est muni d'un moteur d'inférence logique pour raisonner sur ces ontologies, interpréter le contexte et résoudre les conflits. Enfin, il possède un module de protection de la vie privée. Les utilisateurs peuvent définir des règles dans un langage propre à *CoBrA*. L'inférence sur ces règles permet au *context broker* de déterminer s'il peut partager une information donnée. L'idée de Chen est de créer des agents personnels intelligents, ayant accès aux informations personnelles d'un utilisateur. Ces agents communiqueraient avec les *context brokers*. Par exemple, dans l'application « EasyMeeting » imaginée dans [Chen *et al.*, 2004a] pour démontrer la faisabilité de *CoBrA*, l'utilisatrice Alice entre dans une salle de conférence d'un environnement intelligent. Le *context broker* de la salle informe l'agent personnel d'Alice de sa localisation. L'agent personnel ayant accès à l'agenda et sachant qu'Alice donne un séminaire maintenant dans cette pièce, envoie les diapositives d'Alice au *context broker*. Celui-ci contacte l'agent responsable du vidéo-projecteur de la salle pour afficher la présentation d'Alice.

[15] gadgets informatiques
[16] le « courtier en contexte »

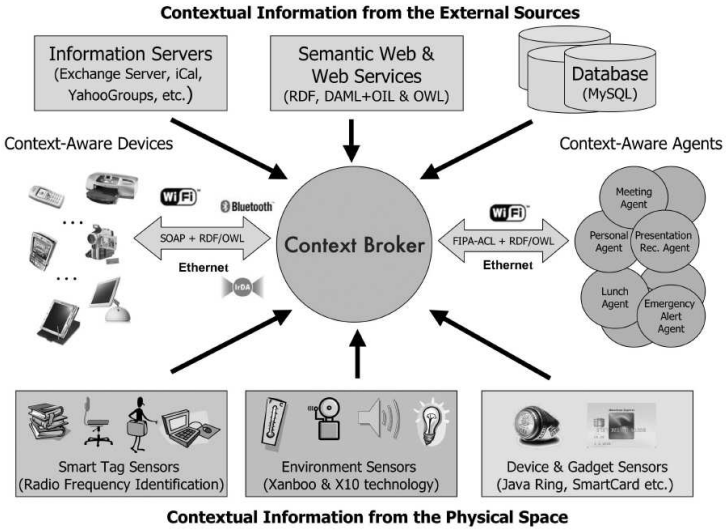

FIG. 2.8 – L'architecture CoBrA, image tirée de [Chen *et al.*, 2004a].

2.2.6.2 Systèmes ubiquitaires en général

[Nurmi et Floréen, 2004] ont réalisé un panorama de ce qu'implique le raisonnement dans les systèmes sensibles au contexte. Pour les auteurs, les données contextuelles peuvent être hiérarchisées. Les données brutes des capteurs sont le contexte de bas niveau et leurs combinaisons forment les contextes de hauts niveaux. Nurmi et al. distinguent ensuite quatre sous-approches au raisonnement sensible au contexte : l'approche bas niveau pour déterminer le contexte de l'utilisateur à partir des données des capteurs ; le point de vue applicatif qui consiste à utiliser le contexte d'une manière intelligente pour l'application ; l'approche à l'écoute du contexte qui détecte les changements de contexte et réagit ; et enfin, l'approche à l'écoute du modèle qui cherche à maintenir les modèles appris dans un état cohérent. Notamment, les auteurs suggèrent l'apprentissage par renforcement pour cette dernière approche.

Un exemple de systèmes ambiants et sensibles au contexte est le système d'exploitation *Gaia* [Roman *et al.*, 2002], qui gère les ressources et les services d'un espace actif. Gaia a été mis en œuvre pour une salle de réunion équipée avec des ordinateurs, des écrans plasma, des projecteurs, des écrans tactiles, des détecteurs de badges, etc. Un utilisateur muni de sa propre session peut entrer dans l'environnement et sa session sera chargée pour utiliser les ressources disponibles. Les appareils mobiles de l'utilisateur sont ajoutés comme ressources à l'espace. L'utilisateur peut accéder à ses fichiers et utiliser de façon transparente toutes les interfaces disponibles. Dans ce système, un contexte est représenté d'une façon analogue à une phrase en anglais, par un prédicat de premier ordre composé du type de contexte, de son sujet, d'un élément de relation et de l'objet. Le sujet est l'entité concernée par le contexte, l'objet est une valeur associée au sujet, et l'élément de relation associe le sujet à l'objet tel un opérateur de comparaison, un verbe ou une proposition. Les règles sont écrites en utilisant la logique du premier ordre sur ces prédicats. Les contextes sont déterminés

par des fournisseurs de contexte basés sur les informations des capteurs. Des applications peuvent ensuite être développées pour ce systèmes d'exploitation ambiant, elles pourront intégrer la prise en compte du contexte de façon native.

Le système *Gaia* a été amélioré en 2004 [Ranganathan *et al.*, 2004] pour y introduire la prise en compte de l'incertitude de la perception du contexte. Le contexte est représenté par un ensemble de prédicats similaire à ce qui a été fait dans [Roman *et al.*, 2002], mais chaque prédicat est désormais complété par un facteur de confiance, une probabilité que le fait exprimé par le prédicat soit effectivement vrai. Le système raisonne ensuite sur ces valeurs de confiance en utilisant la logique floue.

[Assad *et al.*, 2007] ont développé un autre framework distribué pour des services sensibles au contexte et des applications d'ubicomp, appelé *PersonisAD*. Dans le cadre de cet article, le contexte est composé des modèles des entités importantes pour l'application : un modèle de l'utilisateur qui comprend pour l'instant uniquement sa localisation, et les modèles des lieux, capteurs, services et appareils de l'environnement. Ces modèles sont organisés de manière hiérarchique. Les applications collectent des preuves permettant à un composant de raisonnement de déduire des valeurs d'attributs du contexte. Cette valeur peut être envoyée sous différentes formes à différentes applications, choisissant la forme la plus pertinente pour chaque cas. Un composant du système peut être *actif* s'il est associé à une règle qui permettra de le prévenir en cas de changement pertinent du contexte.

Deux applications ont été implémentées pour illustrer le framework *PersonisAD* : *MusicMix* qui joue de la musique correspondant aux préférences de l'ensemble des personnes présentes dans une pièce, et *MyPlace*, qui prévient les personnes présentes des détails pertinents de l'environnement courant. Ce travail propose un cadre pour les environnements intelligents, mais les applications illustrant son utilisation sont minimales et pas très extensibles (préférences et statut utilisateur définis à la main, etc.).

Avec la maturation de l'idée des environnements intelligents est venue l'idée d'*agents* ou *assistants* intelligents qui « personnifient » ou bien orchestrent l'environnement intelligent et avec qui l'utilisateur interagit en priorité. D'après [Richard et Yamada, 2007] et [Bickmore et Mauer, 2006], un agent disponible sur un appareil portatif tel qu'un PDA, étant accessible à l'utilisateur à tout moment, pourrait facilement faire partie de sa vie. De plus, les utilisateurs créent plus naturellement des relations avec un agent animé (d'après [Bickmore et Mauer, 2006], qui ont mené une étude comparant différentes interfaces). Les premiers agents informatiques sensés communiquer avec l'utilisateur et se faire déléguer certaines tâches ont été introduits très tôt par [Negroponte, 1972] et [Kay, 1984], bien avant les environnements intelligents. Cette idée a ensuite été reprise : on peut notamment citer le projet *FRIEND21* [Nonogaki et Ueda, 1991] qui s'est déroulé au début des années 1990 et avait pour but d'étudier les interfaces du 21ème siècle. Les vidéos produites par ce projet[17] illustrent la vie d'une famille accompagnée par des agents intelligents amicaux. Ensuite, les travaux sur des agents intelligents se sont multipliés et nous pouvons commencer par citer ceux de Pattie Maes décrits ci-dessous.

[Maes, 1994] a conduit des travaux au MIT Media Lab sur les agents in-

[17] Ces vidéos sont disponibles à partir de l'URL
http://www.open-video.org/details.php?videoid=8131

telligents. Le but de l'agent décrit dans cet article est de réduire la surcharge de travail et d'information subie par un utilisateur à une époque où de plus en plus de tâches impliquent l'ordinateur. Cet article pré-date l'introduction de l'informatique ubiquitaire ; Il s'agit d'un agent cantonné au bureau informatique de l'utilisateur, mais les principes introduits peuvent s'étendre au cadre de l'ubicomp. Maes emploie la métaphore de l'*assistant personnel* qui collabore avec l'utilisateur et qui devient de plus en plus efficace en apprenant les habitudes, intérêts et préférences de l'utilisateur. L'agent est doté d'un minimum de connaissances initiales et apprend les comportements répétitifs des usagers. L'agent utilise quatre paradigmes d'apprentissage : (a) il enregistre tout ce que fait l'utilisateur et cherche des régularités, (b) il recueille des retours directs et indirects de l'utilisateur, (c) il apprend à partir d'exemple donnés explicitement par l'utilisateur et (d) il peut demander conseil aux agents d'autres utilisateurs, et, avec le temps, il peut même savoir quels agents sont les meilleures sources de suggestions. Au cours de l'interaction, l'agent calcule une prédiction sur l'action à exécuter, avec une valeur de confiance. Il compare cette valeur à deux seuils : « dis-moi » et « fais-le ». Au delà du seuil « fais-le », l'agent exécute directement l'action. Entre les deux seuils, l'agent se contente de proposer l'action à l'utilisateur.

En 2001, [Byun et Cheverst, 2001] ont proposé un assistant personnel sensible au contexte qui exploite un modèle de l'utilisateur. Il s'agit donc de combiner les deux techniques : les modèles d'utilisateurs *(user modelling)* qui permettent de personnaliser les services rendus à l'utilisateur particulier, et l'informatique sensible au contexte *(context-aware computing)* qui permet d'adapter les services à la situation courante. Nous verrons dans la suite (section 2.3) que notre but est également de prendre ces deux aspects en compte, mais dans notre application, le modèle de contexte et le modèle de l'utilisateur sont inclus dans un seul et même modèle.

Les modèles de l'utilisateur ont également été exploités par [Godoy et Amandi, 2005] pour construire des profils des internautes naviguant sur internet. Le profil capture les intérêts et préférences de l'utilisateur pour pouvoir lui suggérer des pages susceptibles de l'intéresser. Les auteurs insistent sur l'importance de rendre ces profils compréhensibles par les internautes pour renforcer leur confiance accordée aux suggestions de l'agent, mais également pour leur permettre de consulter les profils d'autres internautes afin de trouver qui a des goûts similaires aux leurs. Ces profils sont organisés de manière hiérarchique et sont appris au travers des historiques de navigation des usagers. L'apprentissage se fait en intégrant chaque nouvelle expérience dans le profil par un mécanisme de classification.

Les modèles de l'utilisateur consituent un domaine de recherche à part entière *(user modelling)* et, bien qu'ils soient intéressants pour le problème qui nous occupe, nous nous orienterons plutôt du côté des systèmes sensibles au contexte.

Le travail précédent [Godoy et Amandi, 2005] a été poursuivi par [Schiaffino et Amandi, 2006] pour concevoir un agent qui assiste l'utilisateur. Dans cet article, un agent virtuel propose une assistance spécifique au contexte, tout en essayant d'optimiser les interruptions. Comme dans le travail précédemment décrit [Godoy et Amandi, 2005], Schiaffino construit des profils des usagers capturant leurs interactions avec un logiciel en utilisant des règles d'association apprises en observant des interactions réelles de l'utilisateur. Ces règles sont mises à jour incrémentalement, lorsque suffisamment de nouvelles

expériences ont été recueillies. On peut reprocher à cet assistant d'être inopérationnel tant qu'une quantité initiale suffisante d'expériences n'a pas été acquise, et de ne pas avoir d'estimation sur la manière d'agir en cas d'une situation qui n'a encore jamais été observée. L'agent développé aide l'utilisateur à se servir d'un agenda électronique et prend en compte des éléments du contexte tels que la tâche courante de l'utilisateur et sa priorité, l'emploi du temps de l'utilisateur, ses intérêts, buts, préférences, habitudes, contacts personnels et son état émotionnel. Le profil construit comporte deux parties : les préférences d'assistance (composées d'une situation, d'une action de l'agent et d'un facteur de confiance) et les préférences d'interruption (composées de la situation, de la modalité d'assistance et d'un facteur de confiance, complétés optionnellement par la tâche de l'utilisateur, sa pertinence et l'action d'assistance).

[Richard et Yamada, 2007] ont étudié un assistant personnel qui gère les rappels de l'agenda de l'utilisateur. Il essaye de déterminer le moment et la modalité idéaux pour le rappel de chaque événement créé par l'usager, en apprenant les préférences de ce dernier. Leur application TamaCoach agit en tenant compte du contexte et apprend en se basant sur un retour positif ou négatif de l'utilisateur (ces retours étant donnés explicitement et recueillis implicitement). Les éléments considérés sont des informations telles que le délai jusqu'à l'événement, la priorité ou les catégories. L'entrée du système de Richard *et al.* est formée de 28 attributs ayant des valeurs discrètes définies par un ensemble fini de valeurs possibles pour chaque attribut. Le système prend en compte deux types de contexte : son propre contexte d'exécution et le contexte de l'utilisateur. Le contexte du logiciel lui est fourni par le système d'exploitation, il s'agit du nom de l'appareil et de deux indicateurs sur la disponibilité d'une connexion réseau et d'un système de son. Le contexte de l'utilisateur est composé de son état de disponibilité, son humeur, son activité et sa localisation. Ces informations sont fournies directement par l'utilisateur, et non perçues. Ces deux contextes sont pris en compte dans le choix de la modalité du rappel. Le module d'apprentissage est composé de deux systèmes de classeurs XCS[18] en cascade (le premier décidant de la priorité avec laquelle un rappel doit être lancé, et le second – de la modalité) et le renforcement reçu permet de les améliorer. Le système inclut l'utilisation d'états partiellement génériques, ce qui le rend plus général.

[Vallée *et al.*, 2005] présentent un environnement d'AmI qui propose de services contextuels à l'utilisateur en effectuant une composition de services dynamique dans un système multi-agent. Ce travail fait partie du projet européen *Amigo*[19] dont le but était de faciliter la mise en réseau des différents équipements de la maison. L'un des points mis en avant par ce projet est de se concentrer sur l'utilisateur pour qui vivre dans un environnement intelligent pourrait être difficile à première vue. L'environnement produit par *Amigo* est complexe mais sa conception est centrée utilisateur et permet de facilement faire évoluer l'environnement en fonction des exigences changeantes de l'habitant. Cependant, cette évolution n'est pas automatique.

Le concept de maison intelligente *(smart home)* a également été implémenté par [Ricquebourg *et al.*, 2006a, Ricquebourg *et al.*, 2006b]. Les auteurs présentent un système offrant des services en fonction du contexte de l'environnement englobant l'habitant. Le contexte est représenté selon le forma-

[18] Pour plus d'informations sur les systèmes de classeurs, le lecteur pourra se référer à [Sigaud, 2007].

[19] http://www.hitech-projects.com/euprojects/amigo/

lisme des « contexteurs » introduit par [Coutaz et Rey, 2002, Rey *et al.*, 2002, Rey et Coutaz, 2004]. Un « contexteur » est une abstraction logicielle modélisant une relation entre des variables et le contexte observé du système. Les services à fournir selon le contexte actuel sont prédéfinis.

2.3 Apprendre l'intelligence aux environnements ubiquitaires

Comme nous l'avons décrit dans la section 2.1, notre objectif est de fournir aux utilisateurs un environnement intelligent, dans lequel des services transparents, naturels et personnalisés sont proposés. Cet environnement actif est incarné sous la forme d'un *assistant virtuel*.

Comme le décrit [Maes, 1994], les deux problèmes à résoudre pour créer un assistant informatique sont celui de la *compétence* (comment l'assistant acquiert-il la connaissance nécessaire pour pouvoir décider quel service rendre à quel moment) et celui de la *confiance* (comment faire en sorte que l'utilisateur se sente à l'aise en délégant des tâches à l'assistant). Les deux solutions les plus immédiates sont de faire spécifier ces services par le développeur ou bien par l'utilisateur. Pattie Maes montre qu'aucune de ces deux solutions ne satisfait totalement les deux critères de la compétence et de la confiance. En effet, si le développeur prédéfinit tous les services, le problème de confiance n'est pas bien résolu car l'utilisateur aura du mal à maîtriser un système trop complexe et sophistiqué. Si c'est l'utilisateur qui spécifie le fonctionnement de l'assistant, c'est le problème de la compétence qui est mal résolu car il sera difficile pour l'utilisateur de fournir des spécifications englobant tous les aspects d'un système complexe. La solution proposée par Pattie Maes, qui est la solution que nous allons également adopter, est donc d'*apprendre* quels sont les services à rendre dans chaque situation. L'apprentissage a l'avantage supplémentaire d'être très évolutif. Le système s'adaptera aux changements de préférences de l'utilisateur sans que ni celui-ci, ni le développeur n'aient besoin d'intervenir.

[Byun et Cheverst, 2001] estiment également que le mode de vie de l'utilisateur est enclin à changer au cours du temps et que le système doit s'adapter à ces changements. Ceci est aussi l'avis de [Godoy et Amandi, 2005] qui prennent en compte le fait que les intérêts des utilisateurs changent dans le temps. De plus, d'après Maes, il est également important que l'agent puisse donner des explications à l'utilisateur, par exemple de la forme « j'ai pensé que tu voulais faire ceci car la situation était similaire à cette situation passée où tu as fait cette action » ou encore « parce que l'assistant de M. Dupond fait comme ceci et tu sembles partager les habitudes de travail de M. Dupond ». [Assad *et al.*, 2007] ont également intégré la possibilité de fournir des explications à l'utilisateur sur le comportement du système. En addition à cette liste, [Godoy et Amandi, 2005] décrivent également l'importance de pouvoir décrire le modèle à l'utilisateur.

Cette idée d'apprentissage est soutenue notamment par [Remagnino et Foresti, 2005] qui pensent que l'avenir de l'AmI réside dans la recherche en apprentissage automatique. En effet, d'après les auteurs, les modèles pour les systèmes intelligents et les associations entre les sorties des capteurs et un comportement intelligent sont de loin trop complexes pour être câblés manuellement.

En résumé, apprendre automatiquement les préférences de l'utilisateur lui simplifie la tâche, rend le système évolutif et indépendant d'un expert. L'apprentissage que nous mettons en place sera un processus à long terme, intégré au fonctionnement normal de l'environnement *(life long learning)*, et qui prendra donc automatiquement en compte les changements de l'environnement ou des préférences utilisateur.

Notre environnement intelligent est matérialisé sous forme d'un *assistant personnel ambiant*, représenté par le personnage de la figure 2.9. Notre choix d'un personnage dessiné humoristique est guidé par [Richard et Yamada, 2007], qui soutiennent qu'un tel personnage met l'utilisateur à l'aise et l'engage davantage.

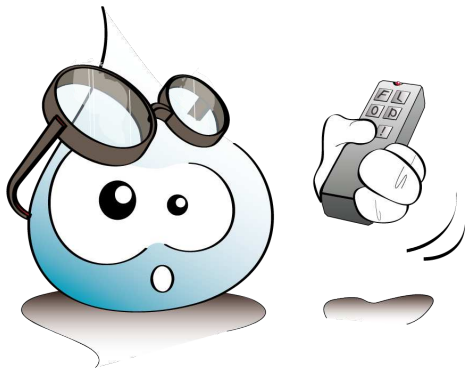

FIG. 2.9 – Le personnage incarnant notre assistant personnel.

2.4 Contraintes

La réalisation de notre système doit prendre en compte plusieurs éléments. Certaines de ces contraintes découlent de l'étude de l'état de l'art menée ci-dessus (section 2.2). D'autres apparaissent du fait que l'apprentissage nécessite un entraînement et que l'utilisateur est celui à qui incombe cette tâche, étant donné que ce sont ses préférences que nous devons apprendre.

(a) Le système ne doit pas être une boîte noire pour l'utilisateur, il doit être intelligible et l'utilisateur doit être en mesure de comprendre son comportement. Le système doit donc pouvoir fournir des explications sur ses actions. Ceci est détaillé dans [Bellotti et Edwards, 2001].

⇒ Le formalisme choisi pour représenter les états de notre système nous permet de respecter cette contrainte. La section 5.6.1 décrit comment nous avons réalisé ceci.

(b) i. L'entraînement doit être simple, non intrusif et peu contraignant ;

ii. Il doit être rapide ;

iii. Le système ne doit pas démarrer à zéro : dans l'état initial, le comportement ne doit pas être totalement incohérent car l'utilisateur refuserait rapidement le système (avant même que celui-ci ait eu le temps de faire ses preuves) ;

⇒ Nous verrons dans le chapitre 5 comment la méthode d'apprentissage choisie répond à ces trois critères.

(c) Le système doit être engageant.

⇒ L'interaction minimale de l'utilisateur avec le système se limite aux retours donnés. Ceci permet au système d'être simple d'utilisation et non intrusif. De surcroît, pour qu'il soit engageant, le système laisse la possibilité à l'utilisateur de spécifier lui-même les actions associées à des situations, soit de sa propre initiative, soit lors d'une phase de questions-réponses, également optionnelle. De plus, le mélange de l'apprentissage (qui est une technique d'intelligence artificielle) avec une configuration manuelle lorsque l'utilisateur le souhaite (ou est d'accord), réduit la difficulté du problème à résoudre.

(d) Le système doit être porteur de valeurs humaines.

⇒ D'abord, ceci est véhiculé par le respect de la vie privée des utilisateurs. Le système ne révèle en aucun cas des informations personnelles, il n'inclut dans son fonctionnement que des appareils enregistrés et il fait la différence entre un appareil privé et public. Ensuite, les valeurs humaines peuvent être véhiculées par les actions proposées. L'on peut imaginer toutes sortes d'actions, notre système est conçu de manière générique par rapport aux actions définies.

(e) L'environnement résultant doit être adapté à l'utilisateur.

⇒ Ceci est sous-entendu car l'environnement apprend les habitudes et préférences de l'utilisateur, il lui est donc adapté dès lors que l'apprentissage est suffisamment avancé.

Ce premier point (a) a été développé par Victoria Bellotti et Keith Edwards de Xerox PARC dans [Bellotti et Edwards, 2001] où les auteurs font valoir qu'un système informatique ne pourra jamais détecter *tous* les aspects du contexte, en particulier les aspects humains. La conception d'un système capable d'agir à la place des utilisateurs doit donc prendre ceci en compte en étant *intelligible* et *responsable* :

> « Context-aware systems that seek to act upon what they infer about context must be able to represent to their users what they know, how they know it, and what they are doing about it. »

Les auteurs mettent en avant quatre points à intégrer dans tout système sensible au contexte : (1) informer l'utilisateur des capacités courantes du système et de sa compréhension du contexte ; (2) pouvoir indiquer à l'utilisateur ce qu'il se passera après une action du système, ce qu'est entrain de faire ce dernier et ce qu'il a fait jusqu'à présent ; (3) révéler à l'usager de l'information sur les autres acteurs ; (4) laisser à l'utilisateur le contrôle des actions le concernant.

En effet, notre assistant prétend suivre les préférences de l'utilisateur alors que celui-ci ne les a pas définies lui-même. Si l'utilisateur ne comprend pas le fonctionnement du système, alors, en cas d'erreur, il risque de le rejeter en considérant qu'il ne fait pas ce que l'utilisateur voudrait. Pourtant, si l'assistant est capable d'expliquer ses erreurs, l'utilisateur sera bien plus enclin à accepter le système et à attendre que les erreurs de comportement se corrigent. Si l'utilisateur comprend le fonctionnement interne de l'assistant, s'il ne considère pas l'assistant comme une boîte noire, alors il accordera une plus grande confiance au système et aura plus de facilité à l'accepter et à l'utiliser.

[Leahu *et al.*, 2008] abordent également cet aspect en parlant de *experience design*. Selon les auteurs, il est fort probable que les systèmes ubicomp ne soient

pas capables de totalement percevoir et interpréter tous les aspects de la situation courante, en particulier le ton émotionnel des usagers. L'idée est de prendre le problème à l'envers et de concevoir les systèmes de sorte que l'interprétation qu'ont les usagers du système comble les lacunes de l'interprétation qu'a le système des usagers (c'est ce à quoi correspond le terme *experience design* : la conception orientée vers l'expérience utilisateur). D'après les auteurs, se concentrer sur la compréhensibilité des systèmes par les humains peut aider à améliorer les problèmes de confiance des utilisateurs dans les systèmes ubicomp.

L'*experience design* a été identifié comme l'un des majeurs défis pour les maisons intelligentes par [Edwards et Grinter, 2001]. Dans cet article, les auteurs pensent qu'un défi pour l'ubicomp est d'aider les occupants à comprendre leurs maisons « accidentellement intelligentes ». Les technologies seront probablement introduites dans les maisons graduellement, par opposition à une maison construite avec un système ubicomp intégré. Il faudra aider les usagers à se familiariser avec les différents appareils, et avec l'ensemble qu'ils forment en étant connectés dans l'environnement intelligent. Il faudra faire comprendre aux usagers ce que l'environnement peut faire, ce qu'il a déjà fait et comment le contrôler. Les auteurs posent la question « comment les usagers vont-ils construire un modèle pour contrôler, utiliser et déboguer des technologies qui interagissent les unes avec les autres dans l'environnement ? Que sera l'expérience d'une maison dans son ensemble lorsque ces technologies sont introduites graduellement, sans les bénéfices d'une conception descendante ? ». [Kozierok et Maes, 1993, Maes et Kozierok, 1993] répondent partiellement à cette question en disant que l'apprentissage étant un processus graduel, il permet de laisser le temps à l'usager de construire un modèle du fonctionnement de l'agent afin de pouvoir prédire ses actions et ainsi lui déléguer des tâches en toute confiance.

Nous en concluons que le paradigme de l'apprentissage est bien une technique intéressante pour résoudre notre problème, à condition de bien respecter toutes les contraintes définies. Ceci sera vérifié dans le chapitre 5.

Enfin, notre but est de faire un système qui se place côté utilisateur dans son intégration du contexte. [Dourish, 2004] distingue deux formes de contexte : le modèle représentationnel et le modèle interactionniste. Le modèle représentationnel est le point de vue classique du programmeur. Il s'agit de déterminer comment un système informatique peut encoder, modéliser, représenter le contexte. De ce point de vue, le contexte est une information sur des aspects de l'environnement dans lequel se déroule une activité. L'autre point de vue est le côté interactionniste qui tente de répondre à la question « comment et pourquoi, au cours de leurs interactions, les personnes atteignent et maintiennent-elles une compréhension commune du contexte de leurs actions ? » Ce point de vue est plus orienté côté utilisateur. Notre approche lie ces deux points de vue. En effet, nous verrons tout au long de ce manuscrit comment l'information de contexte est encodée dans notre système et comment nous fournissons un codage initial. Nous intégrons donc une représentation du contexte. Mais l'utilisateur va modeler le contexte par ses interactions (les retour qu'il fournit pour l'entraînement), ce qui nous place également du côté interactionniste de la vision du contexte.

Pour atteindre cet objectif, nous pouvons identifier plusieurs difficultés qui seront à résoudre.

[Abowd et Mynatt, 2000] parlent des difficultés rencontrées typiquement dans toute recherche en ubicomp. Les auteurs font la remarque que le domaine de

l'ubicomp a souvent affaire à des technologies de pointe, pas toujours robustes et encore mal comprises par les développeurs. Ainsi, les prototypes construits sont eux-mêmes peu robustes et difficilement utilisables par les usagers. Il est donc important pour les chercheurs de construire une histoire fascinante du point de vue de l'utilisateur final. Cette histoire a pour but d'illustrer un système, mais également de servir de base pour évaluer son impact sur la vie quotidienne des usagers.

Il sera difficile pour nous d'avoir un prototype robuste que nous pourrons installer chez les utilisateurs pour le tester. Nous allons remplacer ce test par des évaluations quantitatives en laboratoire sur le système réel (chapitre 6), complétées par une enquête qualitative auprès d'utilisateurs finaux en utilisant une maquette du système (chapitre 3). Comme le suggèrent [Abowd et Mynatt, 2000], ceci nous permettra d'évaluer l'impact du système sur la vie quotidienne des utilisateurs interrogés.

D'autres difficultés résident dans le fait de construire un système fonctionnant dans le monde réel. Il faut prendre en compte les incertitudes, les erreurs de perception, les latences et la complexité du monde réel. Ceci intervient au niveau ingénierie, mais également au niveau conception des algorithmes. En effet, nous souhaitons appliquer les algorithmes classiques d'apprentissage qui sont conçus pour des environnements bien cadrés et maîtrisés.

Par ailleurs, des difficultés sont introduites par rapport au fait que l'utilisateur est dans la boucle de l'apprentissage. Comme il a été expliqué en détail ci-dessus (section 2.3), le système doit être intelligible pour que l'utilisateur l'accepte. Il faut également arriver à équilibrer les interactions avec l'utilisateur, gérer les interruptions, recueillir de lui le retour nécessaire à l'évolution du système.

Appliquer une méthode d'apprentissage *from scratch* (qui démarre à partir de zéro) a pour conséquence le fait que lors de sa mise en route, le système aura un comportement totalement incohérent, ce que l'utilisateur risque fort de rejeter très rapidement. Nous devons donc injecter de la connaissance initiale à notre système afin de lui donner un comportement initial cohérent. Ce comportement sera générique et donc a priori acceptable par tous les utilisateurs. Au fur et à mesure de l'apprentissage, le comportement deviendra personnalisé à l'utilisateur courant. Cet aspect est détaillé section 5.8.

2.5 Exemple d'illustration – premier niveau de précision

Pour finir de présenter et de concrétiser notre problème, nous allons expliciter un scénario exemple de ce que devra faire notre système. Au fur et à mesure de ce manuscrit, nous aurons les éléments nécessaires pour préciser le fonctionnement interne du système réalisant ce scénario.

L'agenda de l'utilisateur déclenche un rappel. L'assistant détecte cet événement et décide de transmettre le rappel à l'utilisateur. Supposons que l'endroit où se trouve celui-ci à ce moment-là ne soit pas connu. L'assistant décide alors de transmettre le rappel par e-mail. Si, par contre, l'utilisateur se trouve à un emplacement connu, l'assistant essaie de prendre contact avec l'utilisateur par sa modalité préférée (message écrit, synthèse vocale, etc.), qui est différente selon

si l'utilisateur est seul ou non. Par exemple, l'utilisateur se trouve seul dans la salle d'expérimentation et sa modalité préférée est l'audio. L'assistant prononce alors le message par synthèse vocale en utilisant les haut-parleurs placés dans cette salle.

Remarque : La question de la gestion des rappels d'événements du calendrier de l'utilisateur a été abordée de manière ciblée dans plusieurs travaux de recherche [Pollack *et al.*, 2003, Shankar et Louis, 2005] et notamment [Richard et Yamada, 2007]. Nous ne cherchons pas à nous mesurer à ces travaux car pour nous, la transmission de rappels n'est qu'une application permettant d'illustrer notre système.

Chapitre 3

Enquête grand public

Avant d'entrer dans les détails de notre méthode, nous allons d'ores et déjà présenter une évaluation qualitative de notre assistant virtuel. Les chapitres 6 et 7 ci-après présenteront des expériences visant à évaluer nos algorithmes d'apprentissage. Ces expériences seront de nature quantitative, elles compareront les résultats numériques des différentes variations de nos algorithmes. Elles mesureront les performances du système. Ici, il s'agit d'évaluer la réaction des utilisateurs finaux d'un tel système. Nous allons aborder la façon dont les utilisateurs perçoivent l'idée même d'un tel assistant ambiant.

3.1 Présentation de l'enquête

Nous avons mené une enquête auprès de diverses personnes pouvant être des utilisateurs de notre assistant. Cette enquête sociologique a été mise en œuvre pour trois équipes du laboratoire LIG : IIHM, MAGMA et PRIMA. Elle a été dirigée par Nadine Mandran et a évalué simultanément deux systèmes distincts mais complémentaires : le nôtre et le système interactif COMPOSE réalisé par Yoann Gabillon, doctorant dans les équipes MAGMA et IIHM du laboratoire LIG, dont le but est de répondre aux requêtes de l'utilisateur en composant des services de manière (semi-)automatique, dynamique et contextuelle [Gabillon *et al.*, 2008]. Nous avons présenté aux utilisateurs ces deux systèmes comme deux parties d'un même assistant : une partie omniprésente en tâche de fond ayant pour but d'accompagner l'usager, de lui faciliter le quotidien (utilisée en situation normale), et une partie que l'usager pourra invoquer lorsqu'il se trouve confronté à un problème ou une question (utilisée en situation d'exception).

3.1.1 Objectif

L'objectif de cette étude était de mesurer les attentes et les besoins des sujets vis-à-vis de l'« informatique ambiante », en situation d'exception et en situation normale. Les sujets ciblés étaient des personnes actives (par opposition aux personnes à autonomie réduite). Le contexte d'utilisation du système n'était pas limité au lieu de travail, mais incluait le domicile, la voiture ou bien des situations telles que « en vacances », « en retard » ou encore « perdu ».

3.1.2 Profil des sujets

Nous avons recruté 26 sujets à interroger. Tous étaient des non-informaticiens. Certaines personnes étaient à l'aise avec un ordinateur et l'utilisaient de manière quotidienne, pour le travail et/ou les loisirs, d'autres étaient des néophytes, mais aucun n'avait de compétences poussées en informatique. La répartition des sujets en groupes d'âge et de sexe est donnée dans le tableau 3.1.

Classe d'âge \ Sexe	Femme	Homme	Total
18-25	3	6	9
26-40	4	3	7
40-60	4	3	7
60+	1	2	3
Total	12	14	26

TAB. 3.1 – Répartition des sujets de l'enquête selon l'âge et le sexe.

Nous avons également équilibré les sujets selon les trois critères récapitulés dans le tableau 3.2.

Lieux d'habitation	urbain / périurbain / rurbain / rural
Nombre d'enfants	avec ou sans jeunes enfants de moins de 7 ans et de moins de 18 ans
Type d'habitat	maison / appartement

TAB. 3.2 – Autres critères de recrutement.

Enfin, le tableau 3.3 fournit la répartition des sujets selon les critères du lieu d'habitation, du statut et de la catégorie professionnelle.

3.1.3 Entretien

Les entretiens suivaient une grille préétablie comportant la description des deux systèmes à évaluer et des questions ouvertes. Cette grille est fournie en annexe A.1. De plus, l'interviewer avait avec lui un ordinateur portable servant à présenter au sujet la maquette des deux systèmes. Cette maquette est réalisée sous la forme d'un document PowerPoint interactif, partiellement fournie dans la section 3.1.4). Chaque entretien durait environ une heure et se déroulait au domicile du sujet ou bien au laboratoire. L'audio des entretiens était enregistré. Une première partie de l'entretien était destinée à évaluer le niveau de compétences informatiques du sujet, mais également à l'interroger sur les nouvelles technologies en général : ses connaissances, sa perception et son usage des nouvelles technologies. Cette partie est également utile pour mettre le sujet à l'aise et le mettre dans le contexte afin qu'il se représente ensuite mieux les deux assistants.

Dans un deuxième temps, l'interviewer présentait au sujet tour à tour les deux maquettes : d'abord notre assistant ambiant, ensuite l'assistant du système COMPOSE. La manière de présenter notre assistant est décrite dans la section 3.1.4. Les questions posées ensuite visaient à connaître l'opinion du sujet

Habitation	
Rurale	3
Rurbaine	5
Urbaine	18
Statut	
Actifs	14
Inactifs	9
Retraités	3
Catégorie professionnelle	
Artistes	1
Professions supérieures	8
Techniciens	4
Employés	2
Ouvriers	2
Étudiants et inactifs	9

Tab. 3.3 – Répartition des sujets selon d'autes critères.

quant à chacun des assistants, les avantages et inconvénients que voyait le sujet, les situations dans lesquelles le sujet trouverait l'assistant particulièrement intéressant, etc. L'interviewer cherchait des réponses à ses questions, mais également à faire s'exprimer le sujet librement à propos des assistants et ce qu'implique leur usage afin de recueillir le plus d'éléments intéressants possible.

L'entretien cherchait également à mesurer l'acceptabilité de notre assistant. Les questions posées à ce propos étaient les suivantes :

- « Si l'assistant apprend mal, autrement dit, s'il n'arrive pas à vous proposer les bons outils au bon moment, quelle sera alors votre réaction ? »
- « Si l'assistant commet des erreurs mais que vous savez qu'il est en train d'apprendre et de s'adapter à votre façon d'agir, quelle serait alors votre réaction et votre avis sur cet objet ? Lui laisseriez-vous une chance ? »
- « Seriez-vous prêt(e) à passer un peu de temps pour répondre aux questions de l'assistant (questions sur ce qu'il a compris de vos habitudes), afin qu'il apprenne plus vite (que son comportement soit plus correct) ? »
- « Si l'assistant était capable d'expliquer ses décisions, qu'est-ce que cela vous apporterait ? »

Enfin, nous nous intéressions au problème de la surveillance par le système, problème auquel l'on fait communément référence comme le problème de « Big Brother ». D'abord, il était intéressant de voir si le sujet abordait ce problème par lui-même. S'il n'y pensait pas, l'interviewer lui posait alors des questions visant à savoir s'il trouverait un tel système fiable, s'il pourrait lui faire confiance et si oui ou non il sentirait son anonymat et sa liberté d'agir mis en péril par ce système.

3.1.4 Présentation de notre assistant

La maquette présentée lors de l'interview suivait un scénario. Elle était réalisée sous la forme d'une document PowerPoint dont chaque diapositive illustrait une étape du scénario. De plus, il s'agissait d'une maquette interactive, l'inter-

viewer devait cliquer sur la diapositive comme s'il s'agissait d'un vrai logiciel afin de poursuivre le scénario. Ces diapositives sont présentées ci-dessous.

FIG. 3.1 – Diapositive 1 de la maquette.

« Il est 8h moins 5 et l'ordinateur portable de l'utilisateur est en mode personnel, le fond d'écran contient une photo de sa famille, les dernières recherches internet sont encore ouvertes : horaires de cinéma et recherche de restaurants. Les icônes personnelles apparaissent sur le bureau. L'assistant (Aladdin) est toujours présent, ainsi qu'un curseur permettant à tout moment de dire si on est satisfait du système (en principe c'est la satisfaction par rapport à la dernière action de l'assistant, mais on sait que l'utilisateur va parfois exprimer une satisfaction plus générale). »

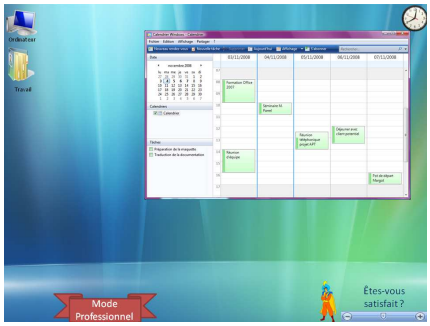

FIG. 3.2 – Diapositive 2 de la maquette.

« À 8h00, l'assistant est passé automatiquement en mode travail. Il a changé le fond d'écran et les icônes, et au lieu des recherches internet personnelles, il a ouvert l'agenda professionnel. »

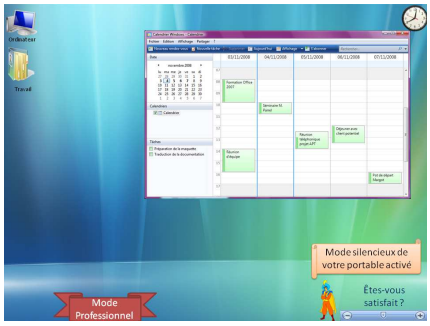

FIG. 3.3 – Diapositive 3 de la maquette.

« Un message apparaît et disparait : *Mode silencieux de votre téléphone portable activé.* Aladdin a automatiquement exécuté cette action. Nous ne sommes pas satisfaits, l'on clique sur le moins du curseur. Celui-ci le place sur le moins puis revient au milieu (il a enregistré le renforcement). Si l'on clique n'importe où dans la diapositive, le scénario est fini, il ne se passe plus rien. Si l'on clique sur Aladdin... »

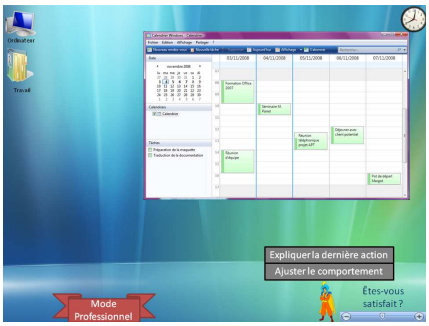

FIG. 3.4 – Diapositive 4 de la maquette.

« Il ouvre un menu avec 2 possibilités :
a) Expliquer la dernière action (passage en mode silencieux) : voir la diapositive suivante ;
b) Ajuster le comportement : permet de choisir un comportement (n'est pas montré dans la maquette, mais propose de choisir une action pour chaque état).
On choisit d'expliquer la dernière action. »

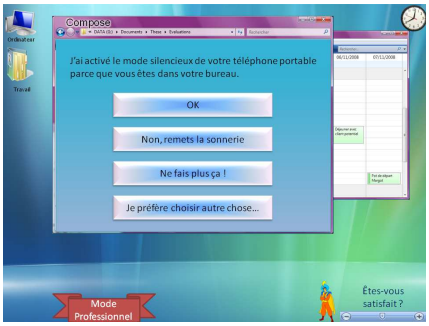

FIG. 3.5 – Diapositive 5 de la maquette.

« Il explique qu'il a fait ça car l'utilisateur est dans le bureau et propose 4 choix.
a) OK : ferme juste cette fenêtre.
b) *Non, remets la sonnerie* : affiche le message *Mode silencieux de votre téléphone portable désactivé.*
c) *Ne fais plus ça !* : désormais Aladdin ne va plus activer le mode silencieux lorsque l'utilisateur arrive au bureau (affiche un message de confirmation).
d) *Je préfère choisir autre chose...* : permet à l'utilisateur de dire quelle sera l'action exécutée lorsqu'il arrive au bureau (choix parmi 3 actions).
Par exemple, on choisit *Je préfère choisir autre chose....* »

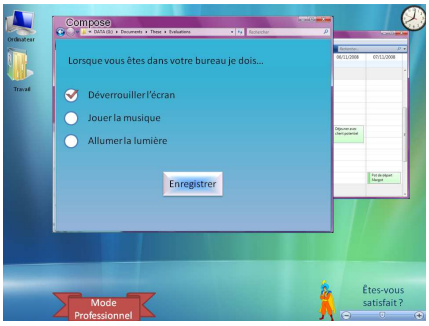

FIG. 3.6 – Diapositive 6 de la maquette.

« L'utilisateur choisit une action qui sera désormais exécutée lorsqu'il entrera dans son bureau parmi les trois proposées par l'assistant (*déverrouiller l'écran*, *jouer la musique* ou bien *allumer la lumière*). On choisit par exemple *déverrouiller l'écran* puis on clique sur *Enregistrer.* »

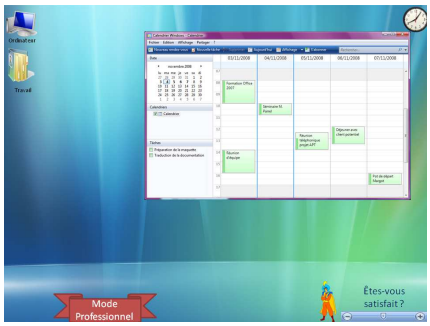

FIG. 3.7 – Diapositive 7 de la maquette.

« Il est 8h05 et on a fini le scénario du mode silencieux, l'utilisateur va travailler. »

FIG. 3.8 – Diapositive 8 de la maquette.

« La journée s'est déroulée ainsi, maintenant il est 18h passé de quelques minutes et Aladdin a remis le mode personnel tel qu'il était le matin. Fin du scénario. »

3.2 Résultats et interprétation

3.2.1 Perception des nouvelles technologies

Les questions préliminaires sur les nouvelles technologies en général ont permis de tirer les conclusions suivantes.

D'abord, le couple ordinateur et internet est perçu avec de nombreux avantages parmi lesquels on peut citer la disponibilité et l'accessibilité de l'information, l'amélioration des pratiques (plus de facilité à réaliser certaines tâches) et un gain de temps.

Ensuite, les inconvénients de ces outils sont également perçus. Parmi eux, on peut citer les problèmes liés à la santé (effets des ondes telles que les ondes WiFi encore inconnus), à l'écologie (gaspillage, trop de déchets électroniques) ou encore à l'addiction (envie d'acheter toujours plus de matériel neuf, des gadgets). D'autres problèmes récurrents sont la rupture du lien social (énormément de choses sont faisables par internet, les gens ont moins de contacts réels en face à face) et le coût de tous ces appareils. D'autre part, l'inconvénient majeur cité par les sujets est le fait que l'on se trouve confronté à de nombreux problèmes techniques : des plantages de logiciels ou des sites internet qui fonctionnent mal. À ceci viennent s'ajouter des problèmes d'utilisation de l'outil informatique et de choix de l'information (tri des résultats d'une recherche internet par exemple). Une résolution par des tiers a été évoquée par plusieurs sujets : en cas de problème, ils préfèrent s'adresser à une personne plus expérimentée.

Nous avons en partie constaté ces points dans l'état de l'art sections 2.2.3 et 2.2.4. D'une part, nous sommes de plus en plus entourés d'appareils divers car ceux-ci sont de plus en plus attrayants (section 2.2.3). D'autre part, il existe encore trop de lacunes dans le fonctionnement de ces systèmes. Trop d'erreurs surviennent et trop souvent, les interfaces ne sont pas adaptées aux utilisateurs (section 2.2.4).

3.2.2 Réactions face à notre assistant

Après la présentation de notre maquette (section 3.1.4), 44% des sujets se sont dits intéressés par le système, 13% ont même été conquis.

Les sujets ont trouvé que l'assistant permet une meilleure organisation et une meilleure gestion du temps. En particulier, la gestion que peut offrir l'assistant est intéressante lorsque les emplois du temps ne sont pas fixes ou encore lorsque la vie professionnelle interfère avec la vie personnelle au cours de la journée. Des femmes qui ont une vie professionnelle active, avec des emplois du temps dynamiques et des enfants à gérer, ont été très intéressées par ce produit. Parmi les réactions des sujets on peut citer :

- « Pour planifier et faire les choses à ma place » ;
- « J'ai plus d'affinité avec mon ordinateur (par opposition au portable), ça fait partie d'un tout » ;
- « Les tâches récurrentes, il m'arrive de ne pas m'en rappeler, le portable est au fond de mon sac, il se décharge » ;
- « Une grosse lacune en organisation, je suis trop tête en l'air » ;
- « J'ai pas un mode fixe comme une femme qui travaille, mon emploi du temps change tous les jours, c'est hyper compliqué, je fais des sonneries sur mon portable : je suis moins stressée, je regarde jamais ma montre » ;
- « Une réunion que j'oublie c'est pas grave, si j'oublie de chercher ma fille, là ça craint, je préfère qu'il me dise clac clac... c'est l'heure d'aller la chercher » ;
- « Il gère le stress ».

Les sujets ont cité des exemples supplémentaires dans lesquels l'assistant leur semble pertinent. Voici ces exemples :

- Pour rappeler des rendez-vous ;
- Pour rappeler les devoirs ;
- Pour gérer les déplacements : « quand je pars, avoir les horaires de train » ;
- Dans les applications domotiques :
  - « Mettre en route une ventilation » ;
  - « Mettre le chauffage quand j'arrive dans la maison, déclencher des lumières, ... » ;
- Dans les voitures, système d'appel et d'assistance : « si je tombe en panne, qu'il appelle le dépanneur le plus proche, ... ou les assurances, pour faire les déclarations ».
- Dans l'utilisation de l'informatique :
  - Lui faire apprendre le schéma de stockage des fichiers : « éviter de perdre des fichiers mal enregistrés à certains endroits » ;
  - Pour lever certaines difficultés techniques en lien avec l'informatique et le multimédia, un assistant technique simple : « Je sais que quand je mets un CD et que je veux le mettre sur mon MP3, j'ai du mal à le faire parce que je ne maîtrise pas l'outil informatique. Ou quand je

> met des photos. Je suis mauvaise. Si j'avais une sorte d'assistant, pas qui contrôle, mais un assistant qui me dirait – que voulez-vous faire ? L'écouter, le mettre sur votre MP3, l'enregistrer ? ... c'est plus dans ce sens que ça m'intéresserait ».

En ce qui concerne les questions sur l'apprentissage de l'assistant, les personnes intéressées sont prêtes à passer du temps là-dessus. Il est intéressant de noter que l'apprentissage apporte un plus au niveau de la fiabilité du système. En effet, l'assistant semble plus fiable car l'apprentissage est fait par l'usager (« oui, c'est fiable c'est moi qui le fais »).

Comme nous l'avons présagé section 2.3, l'apprentissage en mode progressif est un avantage par rapport à un système qu'il faudrait configurer de manière lourde au début de l'utilisation. De plus, l'apprentissage n'est pas gênant car il est progressif (« un peu comme le système de dictée *dragon speaking* : c'est un apprentissage, c'est pas particulièrement gênant, c'est beaucoup plus agréable de le faire petit à petit que de tout programmer d'emblée sur un programme très long. »). Le mode d'entraînement choisi, à savoir l'apprentissage par renforcement qui n'a besoin que de simples récompenses comme entrées, a également été confirmé comme le bon choix : « c'est un système où la correction est rapide, pas trop énervante, un système relativement simple au début pour ne pas saturer » et « c'est juste un clic... dans la mesure où après ça va faciliter mon quotidien et combler un manque chez moi ». Le modèle d'apprentissage doit être simple et clair : « il faut que ça soit simple, si c'est trois quatre questions en trois quatre ligne, qui sont claires... Il faudrait aussi des systèmes de réponses ouvertes », « Si ce sont des questions très précises, je pense qu'il y en aurait beaucoup trop. Il faudrait les gérer par grands thèmes, chacun d'une dizaine de questions maximum, avec quatre à cinq grands thèmes ».

Lorsque l'action choisie par l'assistant ne plaît pas à l'utilisateur, le choix doit être très rapide : « Quand on n'est pas satisfait, il faut pouvoir dire tout de suite si l'on veut changer l'action ou pas – avoir quelque chose qui s'affiche, je l'annule ou je le laisse comme ça ».

Par ailleurs, il ne doit pas y avoir d'interruption dans une situation où la personne ne doit pas être dérangée : « si je suis au travail, en pleine conversation et que ça coupe la conversation, ça va pas le faire. Je le désactive. Cela dépend de l'impact que ça a sur ma vie personnelle. »

Le temps passé par l'utilisateur pour entraîner l'assistant ne doit pas être long, mais il ne faut pas négliger le fait que certaines actions sont moins fréquentes que les autres, et le système doit également les apprendre. Sur ce temps d'apprentissage nous avons eu les réactions suivantes :

- « Une semaine, pas plus » ;
- « Je laisserai un délai mais pas énorme, si au début il se trompe, pourquoi pas, je me prêterai au jeu » ;
- « Pas très longtemps, en quinze jours on ne fait pas forcément tout, trois semaines environ » ;
- « Tous les jours il risque de me proposer une autre offre, un autre service, ça va devenir agaçant ».

Ceci confirme également que le système doit avoir un comportement correct très rapidement (ce qui sera décrit section 5.8).

Il ressort de ces réactions que parfois les utilisateurs souhaitent avoir le moins d'interactions possible avec le système pour l'entraînement, et parfois les utilisateurs souhaitent, au contraire, pouvoir indiquer explicitement à l'assistant le

comportement à adopter (ceci confirme ce que nous a révélé l'état de l'art section 2.2.4 : les systèmes doivent être *engageants*). Une possibilité que nous avons envisagée (section 2.4) mais non encore explorée est de proposer à l'utilisateur un « débriefing ». Il s'agirait d'une phase de questions-réponses sur différentes situations, le comportement actuel de l'assistant et le comportement souhaité. Cette phase pourrait être proposée à intervalles réguliers par le système, mais jamais imposée, ou bien elle serait sollicitée par l'utilisateur. Dans la section 5.8, nous expliquerons que ceci ne contredit pas le système existant d'apprentissage et pourrait être facilement mis en place.

L'entretien cherchait à savoir si les utilisateurs apprécieraient le fait que le système soit capable d'expliquer ses actions. Les sujets ont confirmé que les explications permettent de comprendre les décisions de l'assistant et cela leur permet de garder le contrôle sur ce qu'il fait. Il est indispensable qu'il explique ce qu'il fait pour justifier les erreurs : « s'il ne m'explique pas je le mets à la poubelle, c'est indispensable qu'il m'explique ». Ceci confirme nos propos de la section 2.3, à savoir que ces explications sont indispensables (d'après [Bellotti et Edwards, 2001] entre autres).

Il est très important pour les usagers de garder la liberté de décider des actions que le système doit exécuter. La phase de réaction à un renforcement négatif est donc très importante : « Je lui dirais "ne fais jamais ça", ou alors "fait ça lundi et mardi ne fais jamais ça", il y a la possibilité de décider moi et pas lui, ça me va très bien ».

Les situations absurdes pouvant être provoquées par une des actions de l'assistant ne sont pas un problème : « Pas de problème quand on peut corriger » (mais elles ne doivent pas causer d'interruption grave, comme nous l'avons mentionné plus haut).

Les explications servent à comprendre comment l'outil fonctionne. Mais un fait intéressant auquel nous n'avions pas pensé est ressorti des entretiens : l'assistant peut également permettre à l'usager de découvrir ses propres habitudes et modes de fonctionnement automatiques dont il n'a pas toujours conscience : « pour moi quelque part, ça permet de me rendre compte aussi que même si je ne m'en rend pas compte, je fais toujours les mêmes gestes. Il peut me suggérer des choses, je vais lui dire non. Mais 90% du temps, c'est utile ».

D'autres remarques concernant l'assistant sont également à noter :

- « Limiter le nombre d'interruptions dans la journée » ;
- « Éviter de mettre l'assistant virtuel dans le coin en bas à droite, cela évoque les autres assistants » ;
- « Limiter le nombre de pannes techniques » ;
- « Les pannes sont plus problématiques quand on est habitué à ce genre d'outil » ;
- « Il faut avoir une sauvegarde des données, tout est à refaire quand ça tombe en panne » ;
- « Disposer d'un système de protection de la vie privée : prévoir des verrous pour la confidentialité, pouvoir tout déconnecter facilement » ;
- « C'est un outil pour aider mais est-ce que ça ne peut pas être détourné pour être fliqué pour savoir où je suis, ce que je fais en permanence, dans l'extrême c'est l'idée de *Big Brother* » ;
- « Disposer d'un service de synchronisation » ; [1]

[1] Nous verrons dans la section 4.4 que nous avons envisagé un tel mécanisme avec un profil

- « Comment corriger quand l'ordinateur n'est pas allumé ? » ; [2]
- « Comment il va vieillir, car il fonctionne bien, mais s'il prend un virus et qu'il commence à éteindre alors que je suis en pleine communication... S'il y a une erreur dans le système où il confond tout, ça risque d'être gênant, il faut un moyen facile de le déconnecter » ;
- « Disposer d'un service pour *désapprendre*, d'un service qui gère les exceptions » ;
- « Tout est programmé, un jour on veut faire tout autrement mais le truc il est programmé pour faire comme ça. Ça doit être hyper compliqué de le déprogrammer » ;[3]
- « Par contre, quand il a une exception, il va faire la même action que d'habitude et ça va nous embêter » ;
- « Pouvoir choisir de mettre en route ou non l'assistant ».

En conclusion, cette enquête a d'abord confirmé nos allégations par rapport aux besoins d'un tel système (sections 2.3 et 2.4) : ne pas être trop intrusif (le renforcement doit être facile à donner, les actions ne doivent pas modifier les applications critiques), l'apprentissage doit être rapide et basé sur peu d'exemples. Le fait d'être capable d'expliquer ses actions est critique pour la fiabilité de l'assistant. Enfin, les usagers sont souvent prêts à passer un peu de temps pour entraîner le système, mais ce temps doit rester court et l'entraînement doit rester simple (peu de questions, pas trop de détails). Le problème de la confidentialité a également été soulevé. L'utilisateur doit garder le contrôle sur ses données et veut pouvoir éteindre l'assistant à tout moment à l'aide d'un simple bouton.

Ensuite, l'enquête nous a révélé d'autres craintes liées à ce genre de systèmes : la peur de devenir assisté, de ne plus être capable de réfléchir par soi-même car le système fait trop de choses à la place de l'utilisateur. Le fait de devenir dépendant, d'avoir un nouveau besoin, et d'être par conséquent « perdu » en cas de panne de l'assistant.

utilisateur qui pourrait être exporté et importé d'un environnement intelligent à l'autre (par exemple entre le bureau et la maison). Ce profil serait transporté sur le PDA ou le téléphone portable de l'utilisateur afin d'être toujours à disposition.

[2] En effet, ce n'était pas montré dans la maquette, mais nous prévoyons d'autres modalités pour donner un renforcement.

[3] En effet, le système apprend en permanence et s'adapte aux changements des habitudes et de l'environnement, mais ce n'était pas non plus montré dans la maquette.

Chapitre 4

Système ambiant

Le but de l'informatique ubiquitaire est de créer des espaces dans lesquels le monde numérique et le monde physique sont liés d'une façon naturelle et transparente pour l'utilisateur.

Cet espace ambiant est obtenu en équipant l'environnement de divers dispositifs numériques intégrés dans un réseau cohérent. Le résultat pour l'utilisateur sont de nouvelles possibilités d'interaction et de nouveaux services, le tout accessible d'une manière naturelle et transparente. Pour réaliser cela, un environnement ambiant repose sur des capacités de *perception*, d'*action* et de *communication*. Ses capacités de perception permettent au système ambiant de maintenir un modèle de ses occupants et de leurs activités, mais également de son état physique. Un environnement devient « actif » lorsqu'il est doté d'une capacité d'agir. Ses actions peuvent inclure de simples présentations d'information. Elles peuvent également s'étendre à la capacité de gérer les communications acoustiques et visuelles et à transporter des documents et des matériels. Il peut enfin s'agir d'actions sur d'autres programmes tels que le courrier ou l'agenda électronique. Ces capacités de perception et d'action peuvent inclure toutes sortes de fonctionnalités, telles que la reconnaissance et la synthèse de la parole, l'observation des gestes, l'observation de la manipulation des objets, et l'observation d'interaction des hommes.

Nous allons parler dans ce chapitre de l'architecture logicielle que nous avons mise en place afin de créer un tel environnement.

4.1 État de l'art

4.1.1 Exemples de systèmes existants

Différentes recherches sont menées dans le domaine de l'informatique ambiante. Certains laboratoires disposent d'un environnement complet, domestique ou bien professionnel, afin d'y tester leurs méthodes.

À l'institut technologique de Georgia *(Georgia Institute of Technology)* est installé une maison intelligente (appelée *the Aware Home*[1] [Kidd *et al.*, 1999]), équipée des dernières technologies du commerce et permettant d'évaluer les systèmes développés auprès d'utilisateurs.

[1] http://awarehome.imtc.gatech.edu/

Le projet *MavHome*[2] [Cook *et al.*, 2003] est un projet de maison intelligente des universités de Washington et du Texas. L'approche de ce projet est de voir la maison intelligente comme un agent intelligent qui perçoit son environnement par des capteurs et peut agir à travers ses actionneurs. L'architecture globale est un ensemble d'agents interconnectés, chaque agent étant formé de quatre couches : la couche physique, la couche de communication, la couche d'information et la couche de décision.

Le MIT mène le projet *aire*[3] : *Agent-based Intelligent Reactive Environments*[4]. L'un des axes de recherche de ce projet sont les espaces de travail intelligents. Trois aspects sont étudiés : la prise en compte de la disposition dynamique du mobilier, les algorithmes de perception (dont de vision) permettant d'inférer l'activité et l'observation de la manière dont les usagers utilisent la technologie proposée.

Le système le plus proche de ce que nous avons réalisé est présenté dans [Crowley *et al.*, 2009]. Il s'agit d'une architecture logicielle multi-couches orientée composants qui permet de construire des applications sensibles au contexte utilisant un modèle de situations (décrit par [Crowley *et al.*, 2002, Coutaz *et al.*, 2005, Crowley, 2006] et dans notre section 2.2.5). Le niveau le plus haut de cette architecture sont les services qui interagissent avec l'utilisateur. Ces services s'appuient sur le modèle de situations qui contient les situations possibles dans l'environnement, leurs transitions et les actions qui y sont associées (un exemple de modèle de situations est donné figure 2.7). Afin de maintenir ce modèle à jour, celui-ci s'appuie sur des composants perceptuels, qui, eux, sont basés directement sur les capteurs et effecteurs et qui sont capables d'interpréter leurs sorties et leur fournir une entrée.

4.1.1.1 Les objets communicants

À la base d'un environnement ambiant sont les objets communicants qui le forment. Au début des années 1990, on a pu observer l'émergence de la mobilité avec les téléphones portables, les ordinateurs portables, la couverture WiFi de plus en plus étendue, etc. Ces objets mobiles n'ont fait que continuer de se répandre, aussi parle-t-on de systèmes ubiquitaires depuis les années 2000-2005. La période 2005-2010 est consacrée à rendre ces systèmes ubiquitaires intelligents. En effet, être entouré d'objets mobiles et ayant la capacité de se connecter et de communiquer ne suffit plus. L'intelligence ambiante a pour but d'utiliser ce réseau d'appareils pour rendre un service global aux utilisateurs, faire en sorte que le système composé d'une multitude d'appareils indépendants soit unifié et cohérent. Cette évolution est illustrée figure 4.1.

Les objets communicants ont fait leur apparition vers le début des années 2000, on peut notamment citer parmi les premiers événements autour de ce thème le séminaire SOC'2001 qui a eu lieu à France Télécom R&D à Meylan (France)[5] et la conférence SOC'2003[6]. Ces conférences donnent la définition suivante des objets communicants : *La notion d'« objet communicant » est née dans le milieu des années 1990 et s'est définie comme le croisement de la mi-*

[2] http://ailab.wsu.edu/mavhome/
[3] http://aire.csail.mit.edu/
[4] Des environnements intelligents et réactifs basés sur les agents.
[5] http://pagesperso-orange.fr/see.ds/soc2001.htm
[6] http://www.minatec.com/grenoble-soc/version_fr/index.htm

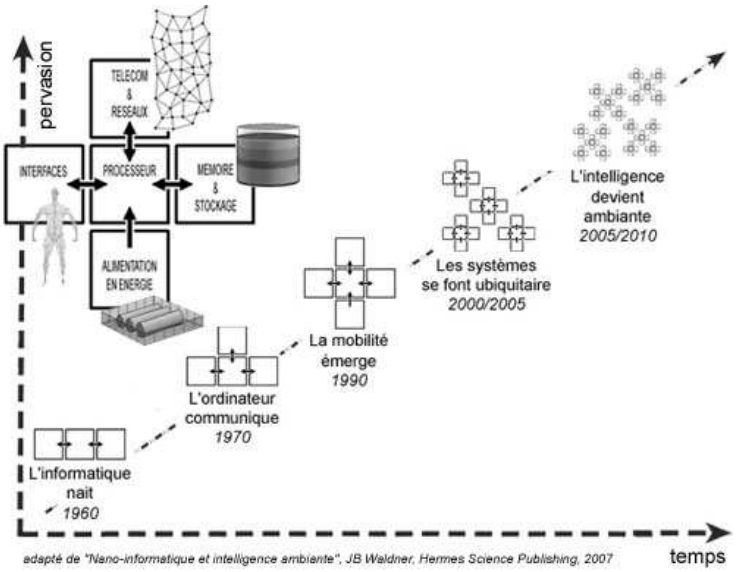

FIG. 4.1 – L'évolution de l'informatique depuis sa naissance jusqu'à aujourd'hui. Source : http://fr.wikipedia.org/wiki/Intelligence_ambiante.

niaturisation des composants et des technologies de traitement de l'information avec le développement des communications sans fil. Le premier séminaire est parti d'un constat sur les évolutions technologiques. D'une part, les terminaux intelligents mobiles (GSM, GRPS et UMTS) et filaires (PCs, terminaux internet, etc.) évoluaient vers le multimédia. D'autre part, les objets multimédia mobiles évoluaient vers la communication et l'intelligence (PDAs, cartes à puce, e-books, appareils photo, etc.). Les *objets communicants* s'apprêtaient alors à envahir notre environnement privé, public et professionnel. Enfin, les coûts de production étaient en baisse et les besoins en mobilité, ubiquité, assistance numérique, simplification de la vie quotidienne, etc. étaient en hausse. Le concept d'objets communicants, en émergence rapide, n'était pas seulement à la convergence des deux mondes « high tech » des terminaux et des objets numériques. Il était également appelé à se diffuser largement dans celui des produits de base de tous types : électroménager, HiFi, vêtements, véhicules, objets personnels divers, etc., selon un concept de *pervasive and ubiquitous computing*.

Mais les objets communicants auront mis du temps avant d'entrer dans le monde du grand public. En effet, au début des années 2000, ces objets étaient encore limités aux laboratoires, par exemple l'armoire à pharmacie intelligente de [Siegemund et Flörkemeier, 2003] ne s'achète pas en supermarché. Aujourd'hui, on voit de plus en plus d'objets communicants faire leur apparition sur le marché et dans la vie quotidienne. Parmi ces objets l'on peut citer *Aibo*[7], le chien robot de *Sony* qui est équipé de plusieurs capteurs tels qu'une caméra et un microphone, qui est relié à internet via le réseau WiFi et qui est doté d'algorithmes de vision par ordinateur et de reconnaissance vocale qui lui permettent, dans une certaine mesure, d'interagir avec son environnement. Mais ce robot est encore trop cher pour vraiment toucher le grand public. Plus récemment sont

[7] http://support.sony-europe.com/aibo/index.asp?language=fr

sorties les nouvelles consoles de jeu, également dotées d'une batterie de capteurs et reliées à internet. Ces objets communicants sont montrés figure 4.2.

FIG. 4.2 – Exemples d'objets communicants connus du grand public. À gauche : Aibo de Sony, au milieu : la console Wii de Nintendo et à droite : la console PS3 de Sony.

Mais l'exemple le plus marquant sont probablement les produits de la marque *violet*[8] dont la devise est *Let All Things Be Connected*[9]. La vision de Olivier Mével et Rafi Haladjian, les fondateurs de cette entreprise, est *celle d'un espace dans lequel la plupart des objets qui nous entourent seraient doués d'intelligence, capables de réagir ou d'interagir avec nous et surtout, connectés au réseau.* Cette vision correspond à celle de Mark Weiser [Weiser, 1991], le père de l'informatique ubiquitaire. À notre connaissance, il s'agit de la première fois que cette vision passe la frontière entre le monde de la recherche et le monde commercial du grand public. En 2005, le premier objet de cette marque apparaissait sur le marché : le *Nabaztag*, un lapin connecté à internet et communicant par des lumières, de la synthèse vocale ou le mouvement de ses oreilles ; *un appareil qui peut fonctionner sans écran et sans clavier ; des objets qui se fondent dans l'environnement de l'utilisateur.* Le pas suivant a été franchi en 2008 avec la sortie des *Ztamp:s*, des tags RFiD sous la forme d'un timbre à coller sur les objets ordinaires entourant l'utilisateur. Le but étant de les approcher d'un lecteur RFiD, le nez du Nabaztag ou bien un objet dédié – *Mir:ror*– qui se branche en USB à un ordinateur, pour exécuter des actions prédéfinies. Ces produits sont montrés figure 4.3. Voici le scénario exemple que l'on peut trouver sur le site internet de la marque :

> « 8h40, vous vous préparez à sortir de chez vous. Sur votre table il y a une auréole de lumière qui respire calmement à côté de votre ordinateur. D'un petit geste vous tendez vos clefs de voiture vers ce mystérieux appareil. Une voix résonne : "aujourd'hui, pluie 14°C". La voix poursuit "vous arriverez en 15 minutes". Sur l'écran de votre ordinateur l'image de la webcam qui filme le principal axe que vous allez emprunter s'affiche tandis que la voix vous lit votre horoscope du jour. Pendant ce temps, vos amis voient apparaître sur votre profil sur les réseaux sociaux "Il est 8h40 je quitte la maison". Au bureau, votre collaborateur favori, reçoit un email disant que vous n'allez pas tarder. Pour finir, comme vous quittez la maison, l'ordinateur se verrouille. »

[8] http://www.violet.net/
[9] Faisons en sorte que tous les objets soient connectés.

Ce scénario rappelle le genre de services que notre assistant veut rendre. Ceci prouve la pertinence de notre recherche. Par contre, les actions exécutées par *Mir:ror* sont prédéfinies par l'utilisateur. Pour chaque timbre collé (ou recollé), il doit donc se rendre sur la page Web correspondante et créer son scénario qui sera déclenché lorsque cet objet sera proche du lecteur. Lorsque l'utilisateur n'est plus satisfait de son scénario, il doit également lui-même aller le modifier.

Notre système pourrait être considéré comme une extension pour ce genre d'appareils. L'utilisateur garderait toujours la possibilité de spécifier son propre scénario, mais l'on peut très bien imaginer rajouter une possibilité de donner un retour au système, afin que celui-ci modifie ses actions. Ceci serait d'autant plus pertinent si ces objets communicants continuent à se banaliser, si les utilisateurs possèdent plusieurs lecteurs et des dizaines de timbres RFiD, la programmation d'une ou plusieurs actions pour chaque timbre et chaque lecteur peut rapidement devenir pénible (d'autant plus que l'ensemble des actions possible peut également devenir de plus en plus vaste).

Figure 4.3: Autres exemples d'objets communicants grand public : les produits de la marque *violet*. À gauche : *Mir:ror*, à droite : le *Nabaztag*.

4.1.1.2 Le réseau pervasif

Les applications telles que les objets communicants décrits ci-dessus (section 4.1.1.1) sont des concrétisations de cette vision du futur qu'est l'informatique ambiante, mais elles n'en traduisent pas l'essence. Afin de pouvoir réaliser la vision de l'informatique ubiquitaire, nous avons besoin d'une infrastructure nouvelle et adaptée au constat qu'autour de chaque utilisateur gravite un ensemble hétérogène d'appareils ayant des capacités de calcul et de connexion. Cette infrastructure commence également à faire son apparition dans le monde grand public (par opposition au monde des laboratoires de recherche) par le biais de l'entreprise *Ozone*[10], dont la vision est de réaliser le *réseau pervasif*, une extension de l'internet tel qu'on le connaît aujourd'hui. Leur vision du futur est un *ordinateur virtuel*, constitué de tous les appareils d'un utilisateur. On retrouve ici l'idée d'une armée d'appareils indépendants et hétérogènes fonctionnant ensemble, collaborant pour constituer une entité unifée, régie par une même logique, appelée « ordinateur virtuel » (illustré par la partie gauche de

[10] http://www.ozone.net/

la figure 4.4). Les appareils d'un utilisateur seraient alors reliés via un micro réseau, un *Personal Network*. Le réseau pervasif est donc le résultat de l'interconnexion de tous ces micro-réseaux personnels (illustré par la partie droite de la figure 4.4). Ce réseau rendrait très facile le développement et la mise en place de services pervasifs. Dans cet environnement, *l'état par défaut des personnes et des objets sera d'être connectés et accessibles et le partage des ressources de l'utilisateur avec d'autres utilisateurs sera courant.*

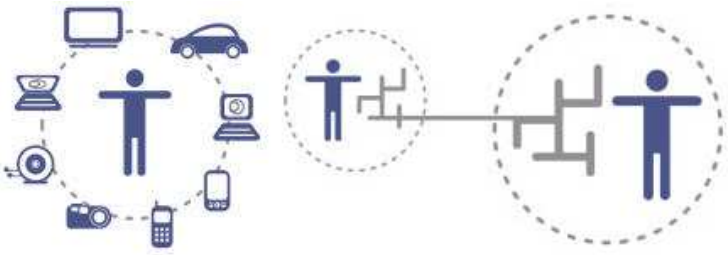

FIG. 4.4 – Illustration de la vision de l'entreprise *Ozone* dont le but est de permettre la réalisation concrète de l'intelligence ambiante. À gauche : l'ordinateur virtuel, à droite : le réseau pervasif.

Le fait que de tels réseaux sont en train d'être développés « dans le monde réel » jusfie et rend d'autant plus pertinente l'étude d'applications pervasives telles que notre assistant personnel.

4.2 Besoins du système

Les choix technologiques que nous allons faire pour construire notre système sont guidés par les besoins rencontrés dans notre cas d'application. Ces besoins sont ceux des système ambiants en général, mais ils sont également orientés de façon à pouvoir réaliser le scénario d'utilisation présenté dans la section 2.5.

Par nature, un environnement ubiquitaire doit intégrer des plates-formes hétérogènes : ordinateurs fonctionnant sous différents systèmes d'exploitation et dispositifs mobiles tels que des téléphones intelligents *(SmartPhones)* ou des PDAs. Par conséquent, une application ambiante est *distribuée*. Cela impose la définition d'un protocole de communication entre les modules qui sont disséminés dans l'environnement. En outre, cela implique la nécessité d'un mécanisme dynamique de découverte de services. Lorsqu'un module a besoin d'utiliser un autre module, par exemple un module de synthèse vocale, il doit d'abord le trouver dans l'environnement.

Cette recherche est généralement assortie de contraintes : le module de synthèse doit, par exemple, être dans la pièce où se trouve l'utilisateur, configuré avec les préférences de l'utilisateur, etc. Nous devons d'abord trouver dynamiquement une machine connectée à des haut-parleurs situés dans cette pièce, et ensuite nous devons trouver le module correspondant sur cet hôte. Enfin, nous demanderons à ce module de se configurer avec les préférences de l'utilisateur, qui portent par exemple sur la voix et le volume sonore. Il est même couramment nécessaire de pouvoir installer et/ou démarrer ce module dynamiquement, s'il n'est pas déjà disponible.

Plus généralement, il est pratique de pouvoir contrôler le déploiement et le cycle de vie des modules. Cet exemple montre que nous avons également besoin d'une base de connaissances sur les infrastructures et les utilisateurs enregistrés. Ce composant sait qu'il y a des haut-parleurs dans la pièce donnée et il connaît le nom d'hôte de l'ordinateur auquel ils sont branchés. Avec la perspective de fournir des services aux utilisateurs, ces connaissances incluent les préférences des utilisateurs telles que leur voix synthétique favorite ou encore leur mode d'interaction privilégié (par exemple l'audio plutôt que la vidéo). Ces connaissances pourraient être distribuées : chaque machine connaît son matériel et ses capacités. Mais nous avons fait le choix d'une base de données centralisée avec une « carte » de l'environnement en termes de matériel. En effet, les requêtes de cette base de données sont très fréquentes et il est plus pratique de regrouper les informations plutôt que de les rechercher sur les différents serveurs. En résumé, nous avons deux types de données : les informations dynamiques fournies par l'annuaire de services sur les modules actuellement en marche d'une part, et d'autre part les informations statiques de la base de données sur les services disponibles en principe, même s'ils ne sont pas exécutés au moment de la recherche. Cette dernière partie n'est généralement pas présente dans les systèmes de l'état de l'art.

Outre les plates-formes, l'hétérogénéité d'un environnement ubiquitaire est également située au niveau des modules qui le composent. Ces modules diffèrent par leur fournisseurs, développeurs et niveau d'implémentation (certains modules peuvent être très proches du matériel, d'autres – de plus haut niveau). Pour ces raisons, les modules peuvent être écrits dans différents langages de programmation. L'architecture ainsi obtenue est souple et évolutive, mais difficile à maintenir. En effet, l'installation et la mise à jour d'un logiciel sur plusieurs machines hétérogènes sont laborieuses. La mise à jour devrait être facile et rapide, sans nécessiter l'arrêt de l'ensemble du système, ou même l'arrêt de la machine en cours d'actualisation. Nous proposons une telle architecture, en réponse à ces besoins avec la combinaison de deux technologies : OMiSCID [Emonet *et al.*, 2006][11] pour la communication et la découverte des services et OSGi (www.osgi.org) pour le déploiement. En outre, nous avons conçu une base de données servant de composant central de connaissances. Cette architecture permet aux développeurs d'applications ambiantes de se détacher de problèmes basiques de mise en œuvre et de se concentrer sur l'intelligence de ces systèmes. Dans la suite (section 4.3), nous détaillons ces aspects.

D'autres systèmes ubiquitaires existent, nous pouvons notamment citer le projet européen *Amigo* [Vallée *et al.*, 2005] dont le but était de faciliter la mise en place de l'informatique ambiante dans les habitats. Pour cela, les partenaires ont développé un intergiciel capable d'intégrer dynamiquement des systèmes hétérogènes afin de fournir une interopérabilité entre divers équipements et services. [Ricquebourg *et al.*, 2006a, Ricquebourg *et al.*, 2006b] présentent également un système ambiant fournissant des services de manière distante à l'utilisateur se trouvant dans son habitat. Ce système utilise la plateforme OSGi que nous présenterons en détail section 4.3.2. L'architecture que nous avons utilisée est propre à l'équipe PRIMA afin de pouvoir plus facilement intégrer les résultats des recherches des différents membres.

[11] *Cf.* également http://omiscid.gforge.inria.fr/

4.3 Architecture du système ambiant

Dans cette section, nous décrivons en détail les technologies sur lesquelles notre architecture est basée et comment l'utiliser pour créer des applications ambiantes.

4.3.1 Communication entre modules : OMiSCID

Comme indiqué ci-dessus (section 4.2), nous avons besoin d'un protocole de communication entre les modules. Pour plus de flexibilité et d'évolutivité, chaque module est un logiciel indépendant et peut être conçu par différents développeurs. Il nous faut un intergiciel *(middleware)* leur permettant de communiquer à travers le réseau. Le contexte d'une application distribuée ubiquitaire introduit certaines contraintes que nous avons évoquées ci-dessus (section 4.2). Nous avons donc choisi un intergiciel spécialement adapté aux applications ubiquitaires : OMiSCID [Emonet *et al.*, 2006].

Hétérogénéité Comme décrit section 4.2, un environnement ubiquitaire est composé de matériel et modules divers (systèmes d'exploitation et langages de programmation variables). L'intergiciel utilisé pour connecter ces modules doit donc être multi-langage et multi-plateforme. OMiSCID est disponible en C++ mais également en Java et en python. L'implémentation Java d'OMiSCID, appelée jOMiSCID [Reignier *et al.*, 2006] est une réécriture complète de la version C++.

Coût réseau Dans un environnement complexe et largement équipé en capteurs, effecteurs et autres dispositifs, les communications entre modules sont très fréquentes et peuvent être chargées en données (images, son). L'utilisateur étant en interaction avec le système, il est important de minimiser la latence. Le surcoût réseau introduit par OMiSCID est minimal : seulement 38 octets d'entête par message.

Robustesse Un système ubiquitaire doit fonctionner en permanence. Un intergiciel centralisé sur un serveur est donc une approche risquée car une panne du serveur entraînerait l'arrêt du système complet. OMiSCID propose une approche décentralisée et donc plus robuste.

Couplage faible Un environnement ubiquitaire est dynamique. Les dispositifs qui le composent peuvent apparaître et disparaître librement et soudainement car certains sont liés aux usagers (par exemple leurs téléphones portables ou PDAs). Ceci implique la nécessité d'un couplage faible entre composants et une découverte dynamique de ces composants. OMiSCID utilise DNS-SD (*Dns service discovery.* http://www.dns-sd.org), aussi appelé Bonjour (cf. Apple) comme répertoire distribué.

OMiSCID est composé de deux couches :

- La communication réseau est gérée par BIP *(Basic Interconnection Protocol)* [Letessier et Vaufreydaz, 2005] qui découpe un flot en un ensemble de messages dont le contenu peut être binaire ou textuel (texte ASCII possiblement formaté en XML).
- La couche services constituée des composants de l'application.

Chaque module est implémenté comme un « *service* OMiSCID ». Il possède un nom unique sur le réseau et se déclare, au démarrage, auprès du domaine. Il peut

ensuite être contacté par n'importe quel autre module dans le même domaine, qu'ils soient exécutés sur la même machine physique ou pas. Ce contact est fait par le biais de « connecteurs » exposés par chaque module. Il existe trois types de connecteurs : *entrée*, *sortie* et *entrée-sortie*. Un service OMiSCID expose également des variables et des constantes (symbole  figure 4.5) représentant son état. Les variables peuvent avoir des permissions en *lecture* (symbole) ou en *lecture-écriture* (symbole ). Un service tiers peut s'abonner aux événements de modification d'une variable. Il est à noter que la spécification des connecteurs et variables d'un service se fait dans un fichier XML (`service.xml`), ce qui facilite également le développement de services.

Les modules peuvent brancher leurs connecteurs afin de communiquer. Le branchement ne peut être fait qu'entre connecteurs « compatibles » : un connecteur entrée (ou entrée-sortie) avec un connecteur sortie (ou entrée-sortie). Naturellement, un connecteur sortie permet d'envoyer un message. Le module décide si ce message est diffusé dans le domaine (message *broadcast*), et alors tous les modules branchés le reçoivent, ou bien envoyé à un module particulier. Symétriquement, un connecteur entrée permet de recevoir des messages à condition d'être branché sur un connecteur sortie. Enfin, un module est prévenu lorsqu'un autre module se branche et se débranche d'un de ses connecteurs.

Les possibilités offertes par cet intergiciel nous permettent de faire communiquer les modules de notre environnement par une approche du type *push/pull*. Un module capteur peut exposer un connecteur sortie sur lequel il envoie des messages lorsqu'il détecte des événements. Par exemple, un module de suivi de personnes peut envoyer un événement chaque fois qu'une personne entre ou quitte sa zone de couverture, un capteur de luminance peut envoyer des événements lorsque la luminance dépasse un certain seuil. Ces événements sont alors diffusés dans le domaine à tous les modules intéressés (qui se sont donc branchés sur le connecteur du capteur). Ceci est l'aspect *push*. Un module peut également envoyer une commande à un effecteur. Pour cela le client branche son connecteur de sortie avec un connecteur d'entrée de l'effecteur et l'utilise pour envoyer une commande. Enfin, deux modules peuvent également communiquer par deux connecteurs entrée-sortie branchés. Le client envoie une requête au serveur, qui répond par le même connecteur. Ceci est l'aspect *pull*.

La figure 4.5 montre les connecteurs de l'assistant personnel (appelé *PersonalAgent*). Le symbole  représente un connecteur entrée-sortie, le symbole  , un connecteur entrée et le symbole  , un connecteur sortie.

Cette architecture permet de connecter différents modules entre-eux, éventuellement développés par différents programmeurs, et d'utiliser chaque module comme une boîte noire, ne connaissant que la convention pour le format de messages (voir la section 4.3.1.1 ci-dessous). De plus, cette architecture facilite le déploiement du système ambiant. En effet, les modules peuvent être exécutés sur différents hôtes, y compris des appareils mobiles : il est possible d'installer OMiSCID et des modules OMiSCID sur un PDA connecté au réseau via le WiFi. En outre, OMiSCID est multi-langage et multi-plateforme puisqu'il est implémenté en C++ pour Windows, Linux et MacOS X, et en Java. Ainsi, les applications ambiantes sont exécutées sur une machine, mais sont ambiantes parce qu'elles ont accès très simplement et de manière transparente à tout autre dispositif mobile ou fixe de l'environnement « équipé » avec un module donnant accès à celui-ci.

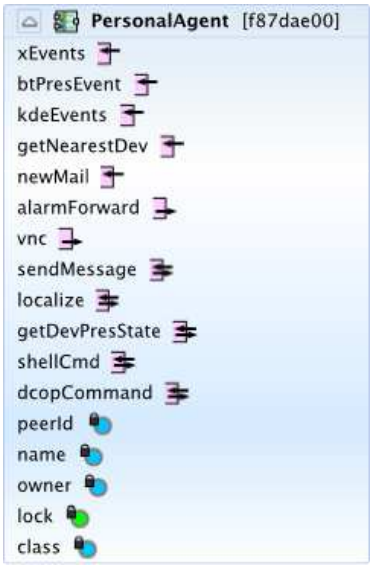

FIG. 4.5 – Le service correspondant à l'assistant personnel, appelé « *PersonalAgent* » et les connecteurs et variables qu'il définit.

4.3.1.1 Format de communication

Afin que les modules puissent se comprendre, nous avons besoin d'une convention pour les messages échangés. Nous avons fait le choix du format XML pour les messages. Chaque connecteur est associé à un schéma XSD (disponible dans la base de données) et les messages d'entrée ou de sortie sont des chaînes de caractères contenant un texte XML valide pour ce schéma. Pour faciliter le traitement du XML, nous utilisons Castor[12] pour un mappage entre le XML et des objets Java. De cette manière, les développeurs ne travaillent qu'avec des objets. Ils n'ont pas besoin de connaître exactement la syntaxe des messages, mais seulement la sémantique (qui devrait être facile à déduire à partir des champs des objets Java).

La figure 4.6 présente un exemple de schéma XSD, il s'agit du schéma décrivant un rappel de l'agenda. La figure 4.7 montre un message XML correspondant à ce schéma. Enfin, la figure 4.8 montre un extrait de la classe Java automatiquement générée par *Castor* à partir du schéma XSD. Dans cette classe, on retrouve les attributs correspondants aux éléments XML (avec leurs accesseurs non représentés sur la figure) et deux méthodes – `marshal` et `unmarshal` – permettant de convertir un objet Java en texte XML et inversement.

4.3.1.2 Exemple

L'assistant personnel doit communiquer un message vocal à l'utilisateur. Ce dernier se trouve dans le bureau j109 de l'environnement. Cette information a été fournie à l'assistant par un module spécialisé capable de déterminer la localisation de l'utilisateur dans l'environnement à tout moment. Ce bureau est équipé de haut-parleurs, l'assistant recherche donc le service « `Text2Speech` » de

[12] http://www.castor.org/ *The Castor Project* de Werner Guttmann

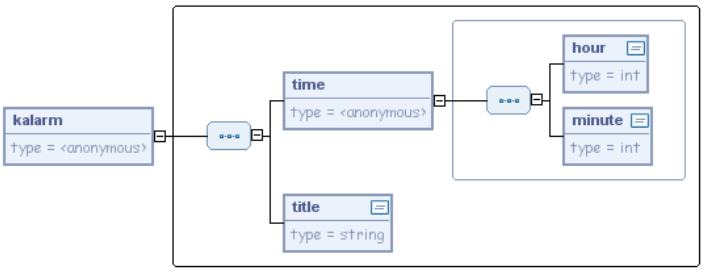

FIG. 4.6 – Exemple : le schéma XSD utilisé pour les événements de rappels de l'agenda.

```
<kalarm valid="true">
   <time hour="14" valid="true" minute="30"/>
   <title>Réunion d'équipe</title>
</kalarm>
```

FIG. 4.7 – Exemple de message formaté en XML selon le schéma de la figure 4.6.

```
package kdeevents ;
public class Kalarm
{
   private kdeevents.Time time ;
   private String title ;
   ...
   public void marshal(java.io.Writer out) {...}
   public static Kalarm unmarshal(java.io.Reader reader) {...}
}
```

FIG. 4.8 – Classe Java générée par *Castor* à partir du schéma XSD de la figure 4.6.

l'unité centrale connectée à ces haut-parleurs. Il branche ensuite un connecteur sortie sur le connecteur entrée de `Text2Speech` et lui envoie le texte à prononcer sous forme d'un message XML correspondant au schéma déclaré. Ce mécanisme est illustré par la figure 4.9.

4.3.1.3 Interface graphique de visualisation des services : OMiSCIDGui

Afin de rendre le développement de services plus confortable, une interface graphique de visualisation des services a été développée par Rémi Emonet. Cette interface permet de visualiser tous les services en cours d'exécution dans un domaine, leurs connecteurs et variables. Elle permet également de se brancher à un connecteur et d'envoyer et recevoir des messages. De même, elle permet d'afficher et de modifier (si les droits d'accès le permettent) la valeur d'une variable. La figure 4.10 montre une capture d'écran de cette interface. En addition,

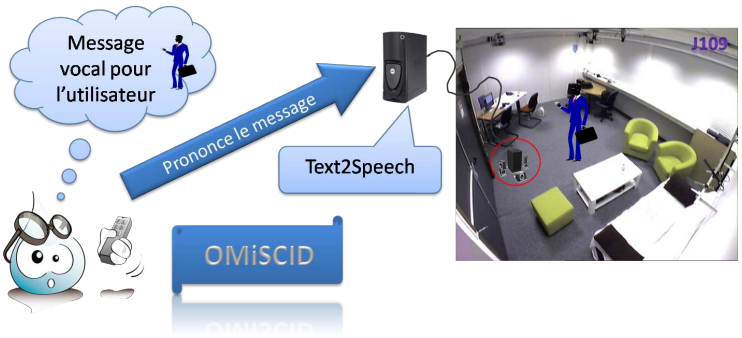

FIG. 4.9 – Exemple de besoin de l'assistant

la figure 4.11 illustre un exemple d'utilisation de cette interface : le suivi des valeurs que prend une variable d'un service. Dans cet exemple, il s'agit d'une variable du service `BluetoothTracker` (décrit section 4.5.1) contenant à chaque instant la liste des périphériques Bluetooth présents dans une pièce. On peut ainsi visualiser l'historique d'occupation de cette pièce.

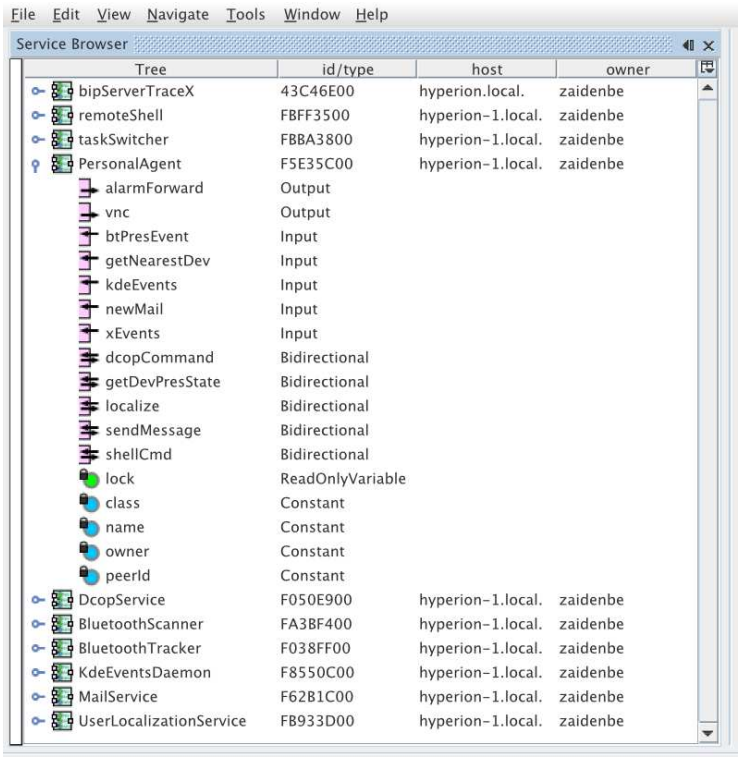

FIG. 4.10 – Une capture d'écran de l'interface graphique de visualisation des services : OMiSCIDGui.

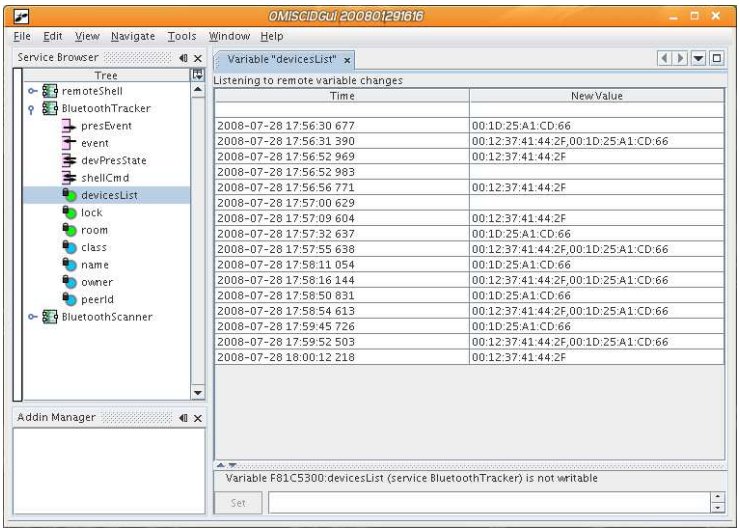

FIG. 4.11 – Une autre capture d'écran de OMiSCIDGui montrant l'affichage des différentes valeurs que prend une variable au cours de l'exécution d'un service.

4.3.2 Déploiement de modules : OSGi

À ce stade, nous avons une architecture distribuée de modules interconnectés. Une telle architecture peut rapidement devenir difficile à maîtriser et à maintenir. Par exemple, un module de suivi de personnes serait installé sur plusieurs machines afin de surveiller plusieurs pièces. Nous avons besoin d'un moyen facile pour installer et pour mettre à jour ce module de suivi sur toutes les machines lorsqu'une nouvelle version est développée.

En outre, nous sommes dans un environnement avec de nombreuses machines fixes, et où des dispositifs mobiles peuvent apparaître dynamiquement. Nous ne pouvons pas envisager l'installation de chaque module manuellement sur chaque machine. Nous avons besoin d'une stratégie opportuniste : n'installer un module sur une machine qu'en cas de besoin. Ceci sous-entend la possibilité de déployer et redéployer à chaud. Par exemple, nous installerons un module de synthèse vocale sur une machine lorsque nous devrons envoyer un message vocal à un utilisateur se trouvant dans la salle où sont les haut-parleurs de cette machine. Le déploiement dynamique des modules se base sur un serveur central contenant tous les modules.

OSGi (*Open Services Gateway initiative*, http://www.osgi.org) fournit ces fonctionnalités. En effet, la plate-forme de services OSGi a été pensée pour pouvoir être facilement administrée à distance, de manière transparente pour l'utilisateur final. Elle rend donc aisée l'installation et la mise à jour distantes de composants, sans nécessiter l'arrêt de la plate-forme. Les modules de l'environnement sont donc mis en œuvre non seulement comme des services OMiSCID, mais à la fois comme des *bundles* OSGi.

Nous utilisons Oscar comme implémentation de OSGi. Une plate-forme Oscar s'exécute sur chaque appareil faisant partie de l'environnement. Au moins deux composants *(bundles)* doivent obligatoirement être installés : un bundle qui intègre jOMiSCID, l'implémentation Java de OMiSCID et un bundle qui in-

tègre Castor, ce qui permet aux modules de décoder les messages qu'ils envoient et reçoivent (section 4.3.1.1). Il est commode d'adapter le fichier de configuration de la plate-forme afin que ces bundles soient installés par défaut. Un dépôt commun situé sur une machine centrale regroupe tous les modules. Il est accessible via le protocole http. Chaque plate-forme Oscar installe ses paquets à partir de ce dépôt et, par conséquent, est liée à celui-ci. Quand un nouveau module est développé, il est ajouté au dépôt et toutes les plates-formes Oscar peuvent simplement installer le paquet correspondant. Quand un module est mis à jour dans le dépôt, toutes les plates-formes Oscar mettent simplement à jour leur bundle. Ceci est très pratique car une fois que cette architecture est en place, les nouveaux modules sont très faciles à installer à chaud. Par défaut, la plate-forme OSGi ne se charge que de l'installation d'un seul bundle à la fois. Si un bundle dépend d'autres bundles pour l'importation de paquetages ou l'usage de services, il est nécessaire d'installer préalablement les bundles fournissant les paquetages ou les services requis. Ces bundles peuvent à leur tour requérir l'installation d'autres bundles et ainsi de suite. Il existe cependant un service dans le dépôt central d'Oscar (OBR – *Oscar Bundle Repository*), le `BundleRepositoryService`, qui se base sur une description des bundles en termes de paquetages importés et exportés et en termes de services requis et fournis pour proposer un déploiement en cascade. Pour mettre à jour un module, il suffit d'exécuter la commande *update* dans Oscar[13].

Afin de réaliser les exemples décrits précédemment (notamment section 4.3.1.2), il faut qu'un module (l'assistant personnel dans l'exemple) soit capable de contrôler le cycle de vie des bundles non seulement sur sa plateforme OSGi locale (ce qui est la seule possibilité offerte par défaut), mais également sur une plateforme distante. Afin de rendre cette opération commode, nous utilisons un module nommé `remoteShell`. Il s'agit d'un service OMiSCID capable d'envoyer des commandes à la plate-forme Oscar sur laquelle il s'exécute. Il expose un connecteur pouvant être utilisé par un autre service OMiSCID souhaitant démarrer, installer ou mettre à jour un bundle sur la même plate-forme que ce `remoteShell`. Ce bundle devrait également être installé par défaut sur toutes les plate-formes Oscar.

L'exemple de la section 4.3.1.2 peut donc être ainsi étendu : l'assistant personnel connaît le nom d'hôte de la machine connectée aux haut-parleurs de la pièce dans laquelle se trouve l'utilisateur. Il recherche le service `Text2Speech` (le module de synthèse vocale) sur cet hôte, mais ne le trouve pas. Il recherche alors le service `remoteShell` (qui doit, en principe, être disponible) sur cet hôte et lui envoie une commande pour démarrer ou installer le bundle voulu. Puis, il attend que `Text2Speech` démarre pour pouvoir s'y brancher et lui envoyer le message à prononcer.

La figure 4.12 présente une capture d'écran de l'interface graphique de Oscar, permettant de visualiser les bundles, de les démarrer, arrêter, mettre à jour et installer/désinstaller.

[13] Il existe même un bundle nommé « File Install » qui surveille un répertoire donné et installe automatiquement tous les bundles lors de leur apparition dans ce répertoire. Il met également à jour automatiquement les bundles lorsque le fichier `.jar` correspondant est modifié dans le répertoire.

FIG. 4.12 – Interface graphique de visualisation et manipulation des bundles OSGi.

4.4 Base de données

Pour réaliser les exemples mentionnés dans les sections précédentes (tels que « envoyer un message vocal à l'utilisateur quelle que soit la pièce dans laquelle il se trouve »), certaines connaissances sur l'infrastructure sont nécessaires. Par exemple, à partir du nom d'un bureau, nous devons déterminer s'il y a des haut-parleurs dans ce bureau et quel est le nom d'hôte de la machine à laquelle ceux-ci sont connectés. Nous avons fait le choix de centraliser ces connaissances. Ces informations sont regroupées dans une base de données disponible à tous les dispositifs, et dont le contenu est alimenté également par tous les dispositifs. Un schéma simplifié de cette base est représenté figure 4.13, montrant les principales tables. Le schéma complet est fourni en annexe, dans la figure B.1.

Cette base de données est un élément essentiel du système ambiant car elle contient des connaissances partagées par tous les modules, et elle contient également un historique de tous les événements importants. Nous verrons dans le chapitre 5 que cette trace est utilisée par les algorithmes d'apprentissage. Cette base de données est donc polyvalente, chacune de ses fonctions étant représentée par un schéma SQL. Nous avons défini quatre schémas : *history*, *infrastructure*, *user* et *services* détaillés ci-dessous.

Schéma *history*

La partie *history*, montrée dans la figure 4.14, stocke tous les événements du système. Les tables `services_history` et `connectors_history` contiennent des événements relatifs aux cycles de vie des modules. Lorsqu'un service OMiSCID est démarré, il enregistre un tuple dans la table `services_history` et un tuple pour chacun de ses connecteurs dans la table `connectors_history`. Le schéma XSD utilisé pour envoyer des messages sur ce connecteur est enregistré. Ainsi, même si ce schéma est modifié au cours du temps, les événements enregistrés dans la base resteront exploitables.

Les tables `events_history` et `actions_history` contiennent tous les événements survenus dans l'environnement (par exemple les entrées et sorties de

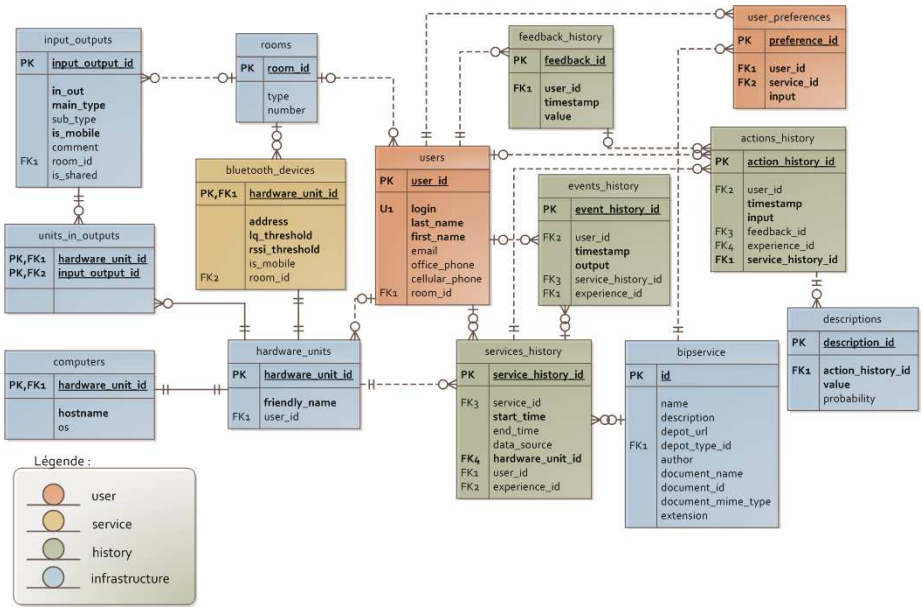

FIG. 4.13 – Les principales tables de la base de données utilisée par l'assistant personnel.

l'utilisateur dans son bureau, les e-mails reçus, etc.) et les actions prises par le système. Ces tables sont utilisées pour l'apprentissage (voir en particulier la section 5.10).

Cette partie est utile pour expliquer à l'utilisateur (si nécessaire) la raison pour laquelle une action a été ou n'a pas été prise. Comme nous l'avons expliqué section 2.3, nous pensons que ces explications permettent à l'utilisateur de faire plus facilement confiance au système.

Enfin, les deux tables `reward_history` et `rl_examples_history` enregistrent les renforcements donnés par l'utilisateur et les états de l'algorithme d'apprentissage par renforcement observés pour l'apprentissage supervisé des modèles de l'environnement dont il sera question section 5.9.

Schéma *infrastructure*

La partie *infrastructure* contient des informations connues et statiques sur l'environnement. Nous pouvons distinguer deux sous-parties sémantiques : la partie « matériels » et la partie « services », présentées dans les figures 4.15 et 4.16.

La figure 4.15 contient les entités physiques de l'environnement enregistrées dans le système : les bureaux (table `rooms`) et le matériel informatique. Nous distinguons les unités centrales des dispositifs d'entrée-sortie qui y sont connectés. En effet, il est important de savoir quel dispositif d'entrée ou de sortie se trouve dans quel bureau (par exemple pour envoyer un message audio ou vidéo à l'utilisateur se trouvant dans un bureau donné), mais il n'est pas important de connaître l'emplacement des unités centrales. Il se peut qu'elles se trouvent dans un local serveur déporté des dispositifs d'entrée-sortie. Il se peut égale-

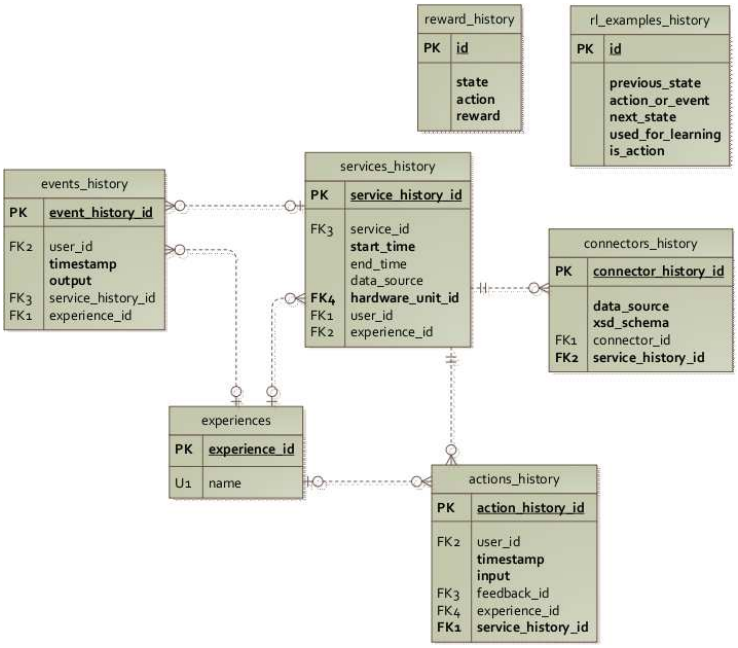

FIG. 4.14 – Le schéma *history* de la base de données.

ment qu'une unité centrale soit connectée à plusieurs dispositifs d'entrée-sortie se trouvant potentiellement dans des bureau différents. Ce schéma de base de données nous permet d'associer chaque dispositif d'interaction avec son emplacement et son unité centrale qui possède un nom d'hôte utilisé pour envoyer des commandes sur ce dispositif.

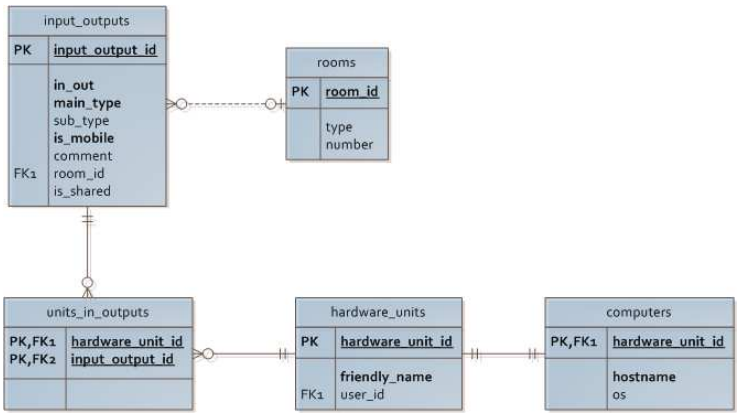

FIG. 4.15 – Le schéma *infrastructure* de la base de données – partie « matériel ».

La figure 4.16 montre la partie de ce schéma relative aux services logiciels. La table centrale est la table `bipservice` représentant les modules de l'environnement intelligent. Les autres tables décrivent les propriétés des modules : leurs connecteurs et variables.

Enfin, nous avons prévu la table `descriptions` pour contenir une description humainement compréhensible pour chaque action entreprise par l'assistant. Cette description serait associée à une probabilité représentant le degré de confiance de l'assistant en cette explication.

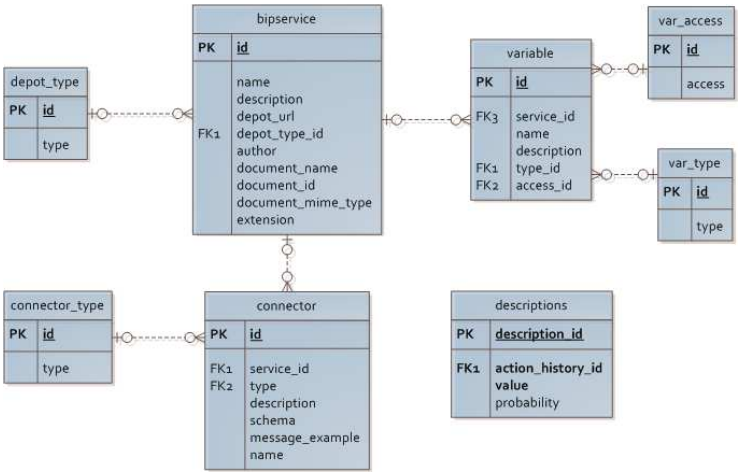

FIG. 4.16 – Le schéma *infrastructure* de la base de données – partie « services ».

Schéma *user*

La partie *user*, montrée figure 4.17, décrit les utilisateurs enregistrés. La table `users` contient les données personnelles telles que les noms de connexion permettant de les identifier sur le réseau local ou encore l'identifiant de leur bureau. Cette table est également utilisée par d'autres tables afin d'associer le matériel et les services aux utilisateurs. La table `user_preferences` peut être utilisée pour stocker explicitement des préférences de l'utilisateur pour l'exécution d'un service (par exemple le choix de la modalité audio pour les rappels). Nous avons prévu cette possibilité mais préférons ne pas nous en servir car toutes les préférences seront apprises (voir le chapitre 5).

Comme évolution future, ces données utilisateur seront disponibles sur le PDA de l'utilisateur comme un profil. Lorsque l'utilisateur entre dans un environnement, son profil est chargé dans le système. Ceci décharge le PDA de tout traitement lourd qui pourrait être nécessaire pour exploiter ces données personnelles (algorithmes d'apprentissage, HMMs, etc.). Le calcul est pris en charge par un autre appareil : un serveur central, l'ordinateur de l'utilisateur, etc. et le PDA, dont les ressources sont très limitées, reste disponible pour son usage premier. Lorsque l'utilisateur quitte l'environnement, il est concevable de migrer le nouveau profil sur le PDA. De cette manière, l'utilisateur dispose sur lui des dernières données s'il en a besoin ailleurs (dans un autre environnement intelligent, par exemple à la maison ou dans sa voiture ou peut-être dans un lieu public).

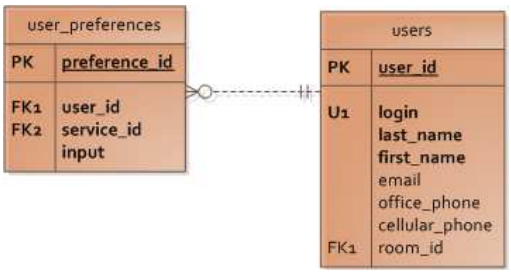

FIG. 4.17 – Le schéma *user* de la base de données.

Schéma *services*

La partie *services*, montrée partiellement figure 4.18, est un terrain libre pour les services additionnels *(plugins)* qui ne sont pas nécessaires au fonctionnement de base du système. Par exemple, si nous disposons d'adaptateurs USB Bluetooth, nous pouvons les utiliser pour détecter la présence d'utilisateurs en détectant la présence de leurs téléphones Bluetooth ou PDAs. Ces informations peuvent être utilisées pour déterminer l'identité d'une cible d'un tracker vidéo. Pour cela, nous disposons de la table `bluetooth_devices`.

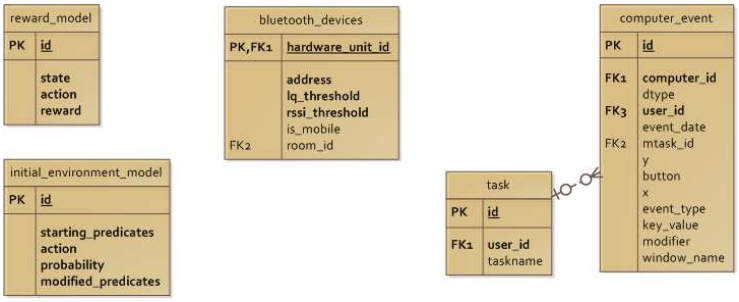

FIG. 4.18 – Une partie du schéma *services* de la base de données.

La table `task` contient les tâches de l'utilisateur. Elles sont définies par l'utilisateur lui-même et c'est également lui qui indique au système sa tâche courante via une simple interface appelée *taskSwitcher*[14] et montrée figure 4.19. Il s'agit de tâches très haut niveau telles que par exemple « écrire un article » ou encore « corriger des TPs ». L'idée est de pouvoir collecter, dans la table `computer_event`, des événements bas niveau associés à la tâche de haut niveau définie par l'utilisateur et de pouvoir ensuite générer une description de la tâche par apprentissage. Ces tâches sont également utilisées dans la partie apprentissage par renforcement du système (*cf.* section 5.6.1).

Les tables `reward_model` et `initial_environment_model` sont utilisées par l'agent d'apprentissage par renforcement (voir en particulier la section 5.10). Plusieurs autres tables sont également présentes et utilisées uniquement par l'agent d'apprentissage, elles seront décrites section 5.6.2.

[14] Développée par Patrick Reignier.

FIG. 4.19 – L'icône *taskSwitcher* est présent dans la zone de notification et permet, d'un clic droit, de changer la tâche courante.

4.4.1 Intégration de la base de données dans le système

Afin de communiquer plus facilement avec la base de données, nous utilisons Hibernate[15]. Nous avons créé un bundle spécifique, incluant Hibernate, que d'autres modules utilisent pour interroger la base de données. Toutes les requêtes utiles sont implémentées dans ce module et les autres modules font simplement des appels de fonctions afin de les exécuter. Ainsi les modules sont indépendants de la base de données. Si la base de données est modifiée, un seul paquet spécifique doit être mis à jour.

La base de données contient les connaissances et la mémoire de l'assistant. Elle lui indique où envoyer une commande pour fournir un service. Elle peut également être utilisée pour expliquer les actions de l'assistant à l'utilisateur. Si la modalité préférée n'a pas été choisie, nous pouvons expliquer, en utilisant la base de données, qu'elle n'était pas disponible dans le bureau donné. Comme la base de données maintient un historique de tous les événements et de toutes les actions, et du cycle de vie de tous les modules, on peut expliquer, par exemple, en cas de défaillance, que cela s'est produit parce qu'un certain module ne fonctionnait pas au bon moment.

4.5 Modules du système ambiant

Nous avons mis en place un système ambiant basé sur l'architecture présentée section 4.3, et utilisant la base de données décrite section 4.4. Nous allons présenter ici les différents modules qui le composent, avant de présenter l'assistant personnel, qui est au cœur de ce système. Les composants présentés ici permettent d'avoir une structure suffisante pour mettre en œuvre l'apprentissage par renforcement et tester nos algorithmes. L'avantage de l'architecture utilisée est d'être facilement extensible. Il est facile de rajouter des modules au système afin de le rendre plus riche, de l'armer de capacités d'action supplémentaires et de capteurs supplémentaires.

Dans cette section, nous n'allons décrire les modules que brièvement. Plus de détails sont donnés en annexe B.2.

4.5.1 Capteurs

Pour plus de détails sur les modules capteurs, le lecteur pourra se référer à l'annexe B.2.1.

[15] *Hibernate Core for Java* de Steve Ebersole, http://www.hibernate.org/

Chaque capteur détecte un type particulier d'événements et diffuse un message sur un connecteur. L'assistant personnel se branche à tous ces connecteurs, reçoit ces événements, et les interprète comme changements de l'état de l'agent d'apprentissage par renforcement (voir la section 5.6.1).

La détection de la présence de personnes dans les pièces se fait en utilisant la technologie Bluetooth. Les ordinateurs fixes des pièces sont équipés de fiches Bluetooth USB et nous détectons la présence des appareils mobiles des utilisateurs (téléphones ou PDAs). En effet, la grande majorité de ces appareils sont aujourd'hui munis de puces Bluetooth et l'hypothèse que les personnes gardent leur téléphone mobile sur eux en permanence est réaliste. Il serait tout à fait envisageable de combiner le détecteur de présence Bluetooth avec d'autres détecteurs, par exemple RFiD, WiFi ou encore vidéo.

La difficulté avec le signal Bluetooth, en particulier le standard Bluetooth 2.0, est sa grande puissance. Un appareil Bluetooth peut être détecté à travers les cloisons des bureaux, à plusieurs mètres de distance. Il ne suffit pas de détecter si un périphérique est présent dans le rayon de portée de la fiche car ça ne signifie pas que le périphérique est bien présent à l'intérieur du bureau, ce qui est l'information intéressante pour nous. De plus, plusieurs fiches Bluetooth dans des bureaux voisins pourraient détecter la présence d'un même périphérique et nous ne saurions pas dans quel bureau se trouve réellement la personne.

Une solution à ce problème est d'utiliser l'information de la puissance du signal. Mais il est difficile de fixer un seuil a priori pour cette puissance. En effet, différents périphériques ont différentes caractéristiques. De plus, le signal Bluetooth admet de fortes variations. La figure 4.20 montre les valeurs de la puissance du signal (appelée « RSSI »[16]) sur une durée d'environ une minute, pour deux dispositifs (un téléphone portable Samsung et un ordinateur portable Dell). Durant l'enregistrement de ces mesures, les deux appareils se trouvaient côte à côte et étaient immobiles. Les variations du signal sont donc inévitables.

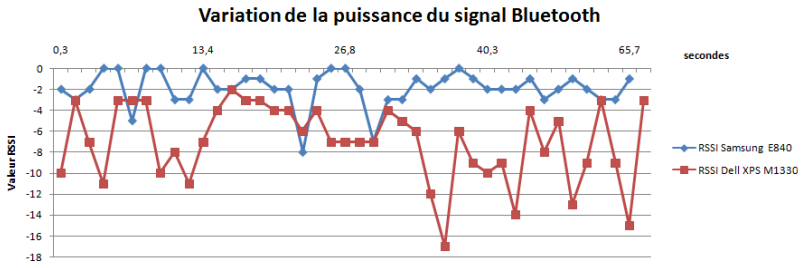

FIG. 4.20 – Mesure de la puissance du signal Bluetooth pendant une minute sur deux appareils.

Ces variations sont en partie introduites par la stratégie d'économie d'énergie des appareils récents[17]. En effet, pour mesurer la puissance du signal, il est obligatoire d'établir une connexion avec le périphérique. Cette connexion est assez gourmande en énergie et les périphériques la coupent au bout de quelques

[16] RSSI est l'abréviation de *Received Signal Strength Indication*, une mesure de la puissance reçue d'un signal radio.

[17] En effet, les téléphones portables d'il y a quelques années seulement n'avaient pas cette stratégie.

secondes. Nous devons donc nous re-connecter en permanence et ceci résulte en un graphe d'autant plus irrégulier.

Au vu de ce graphe, nous constatons qu'il est trop risqué de se contenter d'un seuil fixe car les fausses détections seraient trop nombreuses, ce qui serait pénible pour l'utilisateur.

Pour pallier ces perturbations du signal, nous utilisons des valeurs enregistrées sur une plage de temps de trois secondes[18] et nous calculons la moyenne, ce qui permet de lisser le signal. Cette valeur lissée est ensuite comparée à un seuil afin de détecter la présence ou non du périphérique dans le bureau. La figure 4.21 montre la courbe de cette moyenne calculée pour les deux courbes de la figure 4.20 (qui apparaissent ici en gris). Nous constatons que ces courbes moyennes sont nettement plus stables que les valeurs brutes, et peuvent être seuillées avec un moindre risque de fausse détection.

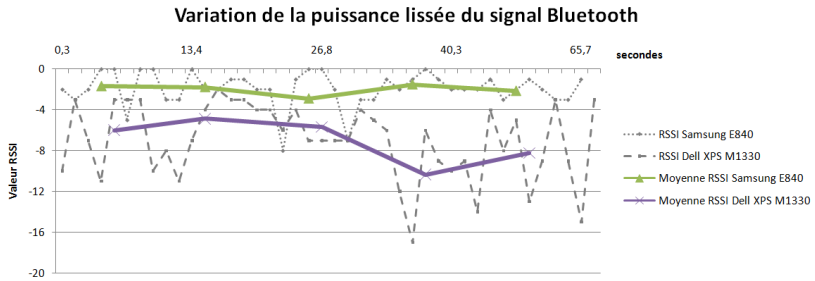

FIG. 4.21 – Mesure de la puissance lissé du signal Bluetooth pendant une minute sur deux appareils.

Le module qui détecte la présence d'un utilisateur dans une pièce est nommé `BluetoothTracker`. Comme son nom l'indique, il suit les périphériques présents dans la zone de couverture de sa fiche Bluetooth. Le tracker ne suit que les dispositifs enregistrés dans la base de données. De plus, il envoie des événements d'entrée et de sortie des périphériques du bureau. Chaque périphérique suivi peut être représenté par une interface graphique (montrée figure 4.22) contenant l'état de présence du périphérique et un bouton permettant de stopper le suivi. L'annexe B.2.1.1 donne des détails supplémentaires sur notre utilisation de la technologie Bluetooth.

Nous disposons par ailleurs d'un service de localisation plus général : `User-LocalizationService` Ce service permet de localiser l'utilisateur dans l'environnement ambiant. Il n'envoie pas d'événements mais répond à une requête de localisation.

Le module `MailService` permet de surveiller l'arrivée de nouveaux messages électroniques dans la boîte de courrier IMAP de l'utilisateur. Il envoie un événement lors d'un nouveau message. Cet événement contient les paramètres du message (destinataire, expéditeur, sujet et corps).

Afin de détecter des événements relatifs à l'ordinateur de l'utilisateur, nous

[18] Cette valeur de trois secondes est empirique et correspond approximativement au temps nécessaire pour « entrer » dans un bureau. De plus, comme il est dit ci-dessus, la durée de la connexion est limitée à quelques secondes. Ce temps permet tout de même d'obtenir un nombre de valeurs suffisant pour que la moyenne ait un sens.

FIG. 4.22 – Boîte de dialogue de suivi d'un périphérique bluetooth, dans le cas où le périphérique est présent (gauche) et absent (droite).

avons implanté le module `KdeEventsService`, spécifique au système de gestion de fenêtres KDE. Toutes les minutes, ce service vérifie l'état des applications qui intéressent l'assistant, c'est-à-dire les rappels de l'agenda, l'état de la musique et de l'économiseur d'écran. Enfin, si besoin, il diffuse un événement.

Enfin, le module `bipServeurTraceX` détecte les événements bas niveau de l'ordinateur de l'usager : les événements clavier et souris, et les changements de fenêtre active. Ce service écoute les événements du serveur X, les filtre et diffuse des événements. Ce filtre permet d'envoyer moins d'événements (on ne s'intéresse pas, par exemple, à chaque position par laquelle la souris est passée, mais au déplacement effectué). En réalité, l'assistant a seulement besoin de savoir s'il y a une activité clavier ou souris, et quelle est la fenêtre active.

Ce service constitue un exemple de la facilité pour utiliser ensemble des applications diverses. En effet, `bipServeurTraceX` est écrit en C++ et n'a pas été développé par la même personne que les autres modules, pourtant il s'intègre parfaitement dans l'architecture de l'assistant personnel.

4.5.2 Effecteurs

Les modules effecteurs que nous avons définis servent principalement à fournir différentes modalités pour contacter l'utilisateur. Nous allons les décrire brièvement ci-dessous et le lecteur intéressé pourra se référer à l'annexe B.2.2 pour plus de détails.

Le premier module intègre un synthétiseur vocal, *Free*TTS[19]. Il prononce « à voix haute » un texte qui lui est transmis à l'aide d'un connecteur entrée. Il est utilisé pour communiquer un message à l'utilisateur où qu'il soit dans l'environnement, pourvu qu'il se trouve à proximité de haut-parleurs par exemple. Grâce à la base de données, on trouve la machine reliée à des haut-parleurs se trouvant dans le même bureau que l'utilisateur, puis, on envoie une requête au service `Text2Speech` qui tourne sur cet hôte. Si ce service n'est pas installé sur cet hôte, on peut même l'installer et le démarrer automatiquement grâce au service `remoteShell`. La lecture en cours peut être interrompue.

Ensuite, nous avons implanté un module fournissant la modalité de communication vidéo : `MessageService`. Ce service permet d'afficher un message écrit sur un écran d'une machine stationnaire ou mobile (on peut ouvrir une fenêtre de message sur un PDA et l'on pourrait également envoyer un SMS sur un téléphone portable, mais nous ne disposons pas des moyens techniques de le faire).

[19] FreeTTS est un synthétiseur vocal gratuit écrit entièrement en Java, http://freetts.sourceforge.net/docs/index.php

Enfin, nous pouvons communiquer avec l'utilisateur en lui envoyant simplement un courrier électronique ; c'est le rôle du service `MailService`. Ceci est utile notamment si la localisation de l'utilisateur est inconnue, et que son PDA n'est pas accessible via le réseau WiFi (l'idéal serait dans ce cas de contacter l'utilisateur sur son téléphone portable, mais nous n'avons pas les moyens techniques pour ce faire). Ce service est à la fois un capteur et un effecteur car il permet non seulement de détecter l'arrivée d'un nouveau mail, mais également d'en envoyer un.

Le dernier service se distingue des précédents car il n'est pas fait directement pour communiquer avec l'utilisateur, mais pour exécuter des actions sur son ordinateur afin de lui rendre des services. Il s'agit du module `DcopService`. Ce service est également à la fois un capteur et un effecteur (la partie capteur est décrite en détail en annexe B.2.1). `DcopService` est un service générique permettant simplement de manipuler des applications du bureau de l'utilisateur. Il est donc possible d'ouvrir des fenêtres, mettre en pause ou jouer de la musique, verrouiller ou déverrouiller l'écran, etc[20].

4.5.3 Assistant personnel

Enfin, l'assistant personnel est implémenté dans un bundle central nommé `PersonalAgent`. Il est connecté à pratiquement tous les modules précédemment décrits, afin d'utiliser leurs services. En se branchant sur les connecteurs sortie des capteurs, il s'abonne à leurs événements. L'assistant sait comment chaque événement modifie l'état du monde. À chaque réception d'événement, l'assistant met alors à jour sa représentation de l'état de l'environnement. Grâce à sa politique, il sait ensuite comment agir en fonction de ce changement. Cette action correspond à un service rendu à l'utilisateur. L'assistant sait également quelle(s) commande(s) à envoyer à quel(s) effecteur(s) pour exécuter l'action choisie dans l'environnement réel.

La figure 4.23 montre tous ces services interconnectés.

4.6 Exemple d'illustration – deuxième niveau de précision

Dans la section 2.5, nous avons présenté un scénario exemple de ce que doit accomplir l'assistant. Après avoir présenté de manière technique notre système ambiant, nous sommes en mesure de préciser en partie ce scénario. La figure 4.24 illustre les explications ci-dessous.

L'agenda de l'utilisateur, géré par le logiciel *KOrganizer*, déclenche un rappel. Le module `KdeEventsService` (décrit en détail section B.2.1.4) est à l'écoute des rappels donc il détecte cet événement. Il envoie un message sur son connecteur de sortie `event`. L'assistant a un connecteur d'entrée (`kdeEvents`) branché à la sortie de `KdeEventsService` : il reçoit cet événement. Il choisit ensuite l'action à exécuter. Supposons que l'action choisie soit de transmettre le rappel à l'utilisateur. Si l'on ne sait pas où se trouve l'utilisateur, la seule façon de

[20] C'est pour cette raison que nous nous sommes concentrés sur KDE dans notre système : l'existence d'un moyen simple pour manipuler les applications du bureau informatique de l'utilisateur.

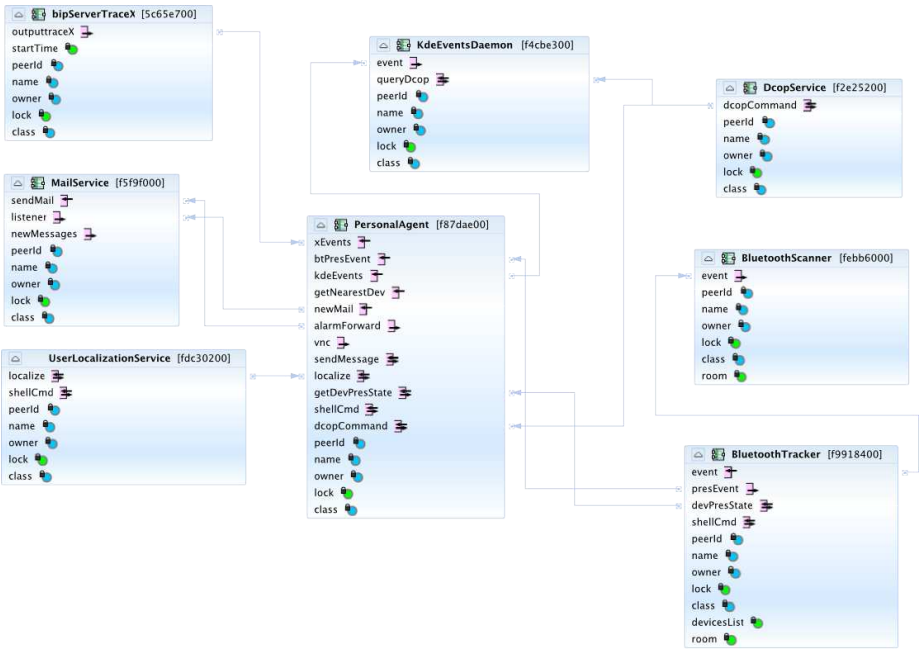

FIG. 4.23 – Vue globale de tous les services et de l'assistant personnel (`PersonalAgent`) les utilisant.

transmettre le rappel est par e-mail[21]. Ainsi, l'assistant entre en contact avec le module e-mail `MailService` (décrit en détail section B.2.1.3) (l'adresse e-mail de l'utilisateur est trouvée dans la base de données) qui envoie un courrier électronique en utilisant *JavaMail*. Si l'utilisateur se trouve à un emplacement connu, l'assistant essaye de prendre contact avec l'utilisateur par sa modalité préférée (message écrit, synthèse vocale, etc.), qui est différente selon si l'utilisateur est seul ou non. L'assistant parcourt les modalités préférées dans l'ordre et interroge la base de données pour savoir si elle peut être fournie dans le bureau où se trouve l'utilisateur.

Pour reprendre le cas de l'exemple décrit section 2.5, l'utilisateur se trouve, seul, dans le bureau J109 (information que l'assistant obtient en interrogeant le module `UserLocalizationService`, décrit en détail section B.2.1.2). Sa modalité préférée est l'audio. La base de données indique qu'il y a bien des haut-parleurs dans ce bureau, et qu'ils sont reliés à l'hôte `protee`. L'assistant recherche, à travers OMiSCID, le module `Text2Speech` (décrit en détail section B.2.2.1) sur cette machine ①. Si ce module n'est pas trouvé ②, l'assistant recherche alors le bundle `remoteShell` (décrit dans la section 4.3.2) sur l'hôte. Si celui-ci n'est pas trouvé, l'assistant ne peut rien faire (il tente alors de transmettre le message à l'utilisateur par un autre moyen)[22]. Une fois `remoteShell` sur la bonne machine trouvé, l'assistant peut alors lui envoyer une requête

[21] Bien sûr, dans l'absolu un message texte sur le téléphone mobile de l'utilisateur serait encore plus approprié, mais nous n'avions pas de moyen technique pour faire cela.

[22] Mais, comme nous l'avons expliqué dans la section 4.3.2, l'idée est que chaque dispositif faisant partie de l'environnement soit équipé d'une plateforme OSGi avec au moins les bundles OMiSCID et `remoteShell` installés par défaut

de démarrage, voire même d'installation, du module initialement recherché ③. `remoteShell` installe le bundle demandé à partir du dépôt central ④ et répond à l'assistant que sa requête a été exécutée ⑤. Ce dernier recherche alors à nouveau le service `Text2Speech` ⑥ et le trouve cette fois-ci ⑦. Il branche alors son connecteur sortie `sendMessage` sur le connecteur entrée `text` du `Text2Speech` de `protee` et lui envoie le message à prononcer ⑧. Si la base de données avait indiqué qu'aucun dispositif audio n'est disponible dans le bureau J109, la procédure aurait été la même pour démarrer et utiliser le service `MessageService`, capable d'afficher un message écrit sur un écran. On peut remarquer que dans cet exemple, l'hôte qui est contacté pour transmettre le rappel pourrait très bien être un dispositif mobile, par exemple le PDA de l'utilisateur, tant qu'il est dans une zone couverte par le WiFi et équipé d'OMiSCID et Oscar.

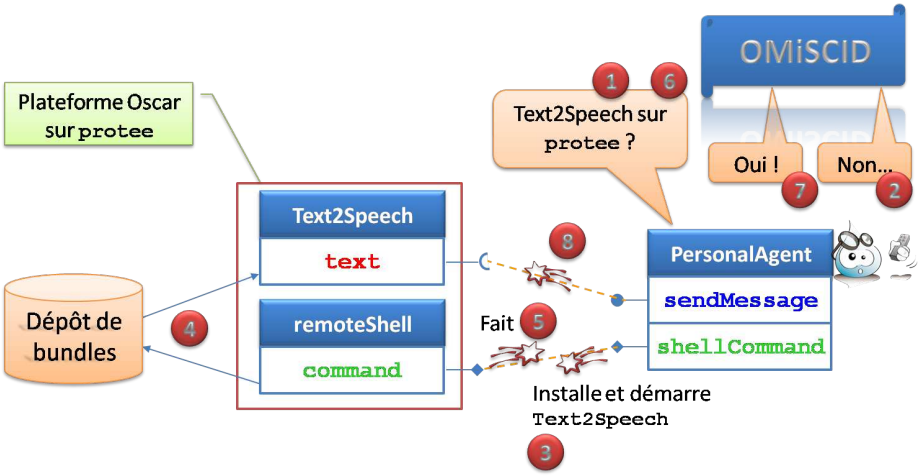

FIG. 4.24 – Illustration du scénario exemple.

Chapitre 5

Application de l'apprentissage par renforcement

Dans le chapitre précédent (chapitre 4), nous avons présenté le système ambiant que nous avons mis en place. Ce système est doté de capteurs et d'effecteurs, ce qui lui permet de percevoir le contexte et d'offrir à l'utilisateur des services appropriés. Notre objectif (présenté section 2.3) est de personnaliser ces services. Après une étude de l'état de l'art, nous avons conclu que la meilleure solution est d'*apprendre* automatiquement ces préférences. Ceci évite à l'utilisateur et au développeur de les spécifier, et permet au système de s'adapter aux futures évolutions car l'apprentissage se fera à vie.

Plusieurs choix s'offrent à nous quant à la méthode d'apprentissage à appliquer. Des travaux précédents ont utilisé l'apprentissage supervisé pour résoudre un problème similaire, notamment [Brdiczka *et al.*, 2007]. Dans cette approche, les auteurs fournissent un ensemble initial de préférences sous la forme de situations et actions associées. L'utilisateur donne un retour vocal ou par le biais de son PDA. Ce retour, et l'ensemble initial, sont l'entrée de l'apprentissage supervisé. Au préalable, une étape de division de situations est effectuée, ce qui permet de rendre les préférences plus spécifiques. Un ensemble d'outils d'apprentissage (SVMs, classifieurs bayésiens, arbres de décisions, etc.) est utilisé pour déduire les rôles et situations à partir de données brutes des capteurs. Toutefois, cette méthode ne répond pas entièrement au critère (a) défini dans nos objectifs (section 2.3) car certaines parties de cet apprentissage ne sont pas humainement compréhensibles (en particulier le résultat des SVMs). Le critère (bi) est également insatisfait car le processus implique l'utilisateur fréquemment et de manière lourde. L'intervention humaine permet finalement de rendre le système plus compréhensible, mais au prix d'un effort important. Notamment, les situations créées automatiquement par division ne possèdent pas d'étiquette lisible, sauf si l'utilisateur exécute une phase d'annotation. Il n'est donc pas possible de présenter le modèle des préférences à l'utilisateur afin qu'il comprenne le fonctionnement interne du système, sans lui demander au préalable de fournir un effort conséquent. Cette approche est donc très lourde et contraignante à mettre en œuvre. Par ailleurs, elle se place du côté du programmeur, alors que notre

but est de nous rapprocher du côté de l'utilisateur. Enfin, il n'est pas possible de faire un apprentissage à vie *(life long learning)* supervisé.

Une autre possibilité est l'*apprentissage par renforcement*. En effet, le problème peut se modéliser comme un processus décisionnel de Markov et il est possible de répondre à tous les critères définis section 2.3. En effet :

(a) L'entraînement en apprentissage par renforcement demande un minimum d'effort à l'utilisateur qui se contente de donner une appréciation positive ou négative lorsqu'il le désire. Il est même possible de recueillir ce retour de manière indirecte.

(b) Normalement l'apprentissage par renforcement est très lent, ce qui contredit le critère (bii), mais nous pouvons l'adapter en trouvant des techniques pour l'accélérer (telles que l'apprentissage par renforcement indirect).

(c) Nous pouvons partir avec un comportement initial par défaut, défini par le développeur ou un expert. Ce comportement sera acceptable pour tous les utilisateurs, puis il sera personnalisé au fur et à mesure de l'apprentissage.

(d) Il faut définir des états et actions du processus décisionnel de Markov de manière humainement lisible. Ceci n'est pas un problème avec l'apprentissage par renforcement car il peut utiliser n'importe quelle forme d'états et d'actions.

Pour ces raisons, nous avons choisi d'appliquer cette méthode. Nous devrons prendre en considération le fait que les algorithmes classiques en apprentissage par renforcement sont conçus pour des environnements bien cadrés et maîtrisés. Ces algorithmes sont souvent appliqués pour apprendre la stratégie de jeux tels qu'un labyrinthe [Morihiro *et al.*, 2004] ou le jeu de go [Silver *et al.*, 2007]. Nous travaillons dans le monde réel. Il faut prendre en compte les incertitudes, les erreurs de perception, les latences et la complexité du monde réel.

Dans la suite, nous allons présenter l'apprentissage par renforcement de manière plus poussée (section 5.1). Puis nous allons établir un état de l'art en apprentissage par renforcement dans le cadre de l'informatique ubiquitaire (section 5.1.9) avant d'expliquer comment nous l'avons appliqué à notre problème (section 5.5).

5.1 Apprentissage par renforcement

> « L'apprentissage par renforcement est une approche informatique de l'apprentissage dans laquelle un agent essaie de maximiser le montant total de la récompense qu'il reçoit en interagissant avec un environnement complexe et incertain » [Sutton et Barto, 1998].

L'objectif de l'agent est d'apprendre, à partir d'expériences, ce qu'il convient de faire en différentes situations. Les récompenses numériques reçues pour les essais sont utilisées dans ce but. Plusieurs algorithmes existent pour effectuer cet apprentissage ; ils se basent tous sur une représentation du problème sous forme d'un *processus décisionnel de Markov*.

5.1.1 Processus décisionnel de Markov

Un processus décisionnel de Markov (qui pourra être noté PDM dans la suite) ou *Markov Decision Process (*MDP*)* est un modèle stochastique permet-

tant de prendre des décisions dans un environnement lorsque l'état du système est connu, mais l'effet des actions est incertain. L'agent modélisé comme un PDM a un certain objectif à atteindre. À chaque étape, il agit sans connaître à l'avance l'état dans lequel cette action va le mener. Il reçoit ensuite une récompense, positive ou négative. Ce fonctionnement est illustré par le schéma de la figure 5.1.

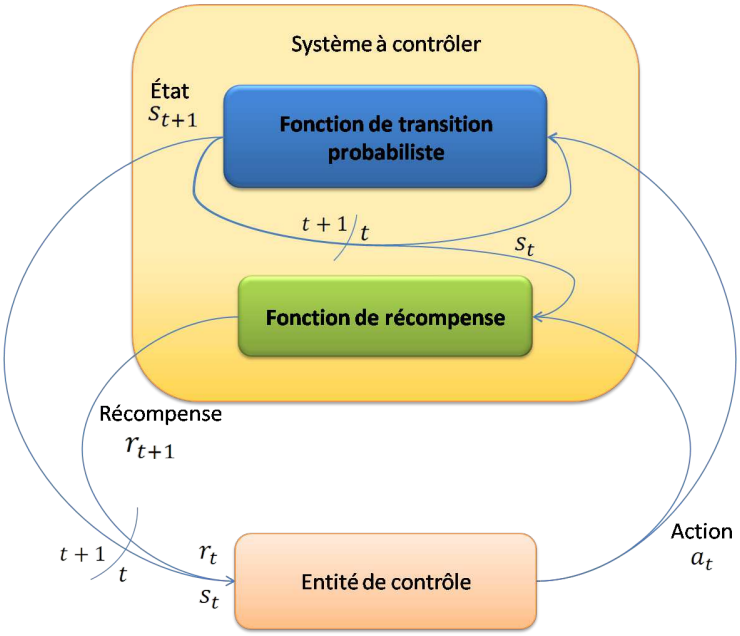

FIG. 5.1 – Cycle de contrôle d'un processus décisionnel de Markov.

5.1.1.1 Définition formelle

Un PDM est défini par un tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ où

- $\mathcal{S}$ est un ensemble fini d'états discrets ;
- $\mathcal{A}$ est un ensemble fini d'actions discrètes ;
- $\mathcal{P}$ est la fonction de transition stochastique markovienne, décrivant la dynamique de l'environnement :

$$\begin{aligned} \mathcal{P} \ : \ & \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,\, 1] \\ & (s,\, a,\, s') \mapsto \mathcal{P}(s,\, a,\, s') = p(s_{t+1} = s' \,|\, s_t = s,\, a_t = a) \end{aligned}$$

- $\mathcal{R}$ est la fonction de récompense immédiate représentant la récompense obtenue après avoir exécuté l'action a dans l'état s :

$$\begin{aligned} \mathcal{R} \ : \ & \mathcal{S} \times \mathcal{A} \to \mathbb{R} \\ & (s,\, a) \mapsto \mathcal{R}(s,\, a) = E\left[r_t \,|\, s_t = s,\, a_t = a\right] \end{aligned}$$

La notation $E\,[.]$ représente l'espérance mathématique (la valeur moyenne).

Un PDM obéit à l'hypothèse de Markov, c'est-à-dire qu'il suppose l'état suivant du système uniquement dépendant de l'état et action courants, et non d'autres états et actions antérieurs. Ceci se formalise par l'égalité :

$$p(s_{t+1} \mid s_t,\, a_t,\, s_{t-1},\, a_{t-1},\, \ldots,\, s_0,\, a_0) = p(s_{t+1} \mid s_t,\, a_t) \tag{5.1}$$

Nous nous intéresserons à des PDM à horizon infini (sans limite temporelle ni état mettant fin à l'évolution du système) et, dans un premier temps, homogènes[1] ($\mathcal{P}$ et $\mathcal{R}$ sont indépendants du temps).

5.1.1.2 Exemple d'un PDM

Le problème du labyrinthe peut aisément être modélisé sous forme d'un PDM. L'espace d'états $\mathcal{S}$ est constitué des 22 cases blanches de la figure 5.2, les cases bleues étant des obstacles.

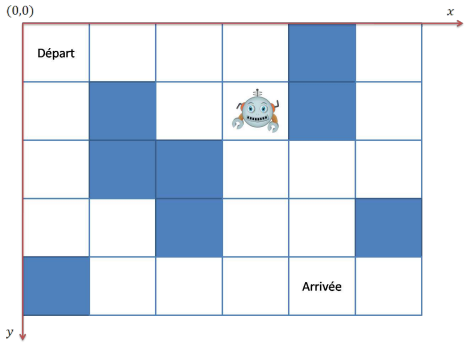

FIG. 5.2 – Exemple d'un labyrinthe simple.

L'espace d'actions $\mathcal{A}$ est constitué de quatre actions correspondant aux quatre directions de déplacement possibles pour le robot : haut, bas, gauche et droite. La fonction de transition $\mathcal{P}$ est stochastique si l'issue d'une action est incertaine. Par exemple, dans le cas d'un robot réel, il y a une probabilité non nulle qu'une commande ne s'exécute pas comme prévu. Par exemple, pour la position du robot sur la figue 5.2, $\mathcal{P}$ pourrait être de la forme :

$$\begin{aligned}
\mathcal{P}((3,1), \text{bas}, (3,2)) &= 0.8 \\
\mathcal{P}((3,1), \text{bas}, (2,1)) &= 0.1 \\
\mathcal{P}((3,1), \text{bas}, (3,0)) &= 0.1
\end{aligned}$$

La fonction de récompense $\mathcal{R}$ serait dans ce cas très simple, elle renverrait 1 pour tous les couples état-action qui mènent à la case d'arrivée, et 0 pour tous les autres.

5.1.2 Politique

L'agent modélisé par un PDM doit acquérir un *comportement*, c'est-à-dire un moyen de décider de l'action à exécuter lorsqu'il se trouve dans un état. C'est le rôle de la *politique*.

[1] ou *stationnaires*.

On définit une politique π comme une application :

$$\begin{aligned} \pi \ : \ & \mathcal{S} \times \mathcal{A} \to [0,\, 1] \\ & (s,\, a) \mapsto \pi(s,\, a) = p(a_t = a \,|\, s_t = s) \end{aligned}$$

Par la suite, on utilisera également la notion de politique déterministe qui associe une action à un état (et non une probabilité d'émission des actions pour chaque état). On utilisera alors la définition suivante :

$$\begin{aligned} \pi \ : \ & \mathcal{S} \to \mathcal{A} \\ & s \mapsto \pi(s) = a \end{aligned}$$

Elle permet de choisir une action a dans un état s. On remarquera que ceci définit une politique markovienne étant donné que la décision (le choix de l'action) ne dépend que de l'état courant et non des états antérieurs.

L'agent est chargé de la construction d'une politique markovienne optimale π^* qui devrait maximiser l'espérance de R_0, c'est-à-dire la somme des futures récompenses espérées actualisées sur un horizon infini, R_t étant définie comme suit :

$$R_t = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k} \tag{5.2}$$

où $\gamma \in [0,\, 1[$ est le *facteur d'actualisation* (également appelé *facteur de dépréciation*) et représente le poids attribué aux récompenses futures, et $r_t = \mathcal{R}(s_t,\, a_t)$ est la récompense obtenue à l'instant t.

Qui dit politique optimale doit définir un critère d'optimalité, une fonction d'utilité à optimiser. Il s'agit de la fonction de valeur décrite dans la section suivante (5.1.3).

5.1.3 Fonctions de valeur

Nous cherchons à trouver une politique optimale π^* maximisant la somme des futures récompenses espérées actualisées sur un horizon infini. Cette espérance est la *fonction de valeur* de la politique, notée $V_\pi(s)$[2] où $s \in \mathcal{S}$ est l'état initial. $V_\pi(s)$ est donc la quantité de récompense que l'agent peut espérer obtenir en partant de l'état s puis en suivant la politique π :

$$\begin{aligned} V_\pi(s) &= E_\pi[R_t \,|\, s_t = s] \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k} \,|\, s_0 = s \right] \end{aligned} \tag{5.3}$$

où s_0 est l'état initial.

[2] $V^*(s)$ pour la politique optimale.

De la même façon nous pouvons définir la valeur d'une action a exécutée dans un état s (autrement dit la *qualité* du couple (s, a)) dans le cadre d'une politique π, notée $Q_\pi(s, a)$ et définie par l'équation suivante :

$$\begin{aligned} Q_\pi(s, a) &= E_\pi[R_t \mid s_t = s] \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k} \mid s_0 = s,\ a_0 = a \right] \end{aligned} \tag{5.4}$$

Il s'agit cette fois de la quantité de récompense que l'agent peut espérer obtenir en partant de l'état s, exécutant l'action a puis en suivant la politique π. Cette fonction Q est appelée *fonction de valeur d'action* ou encore Q-*fonction*. De manière similaire nous parlerons de Q-*valeur* d'un couple (s, a) pour désigner la quantité $Q_\pi(s, a)$[3].

Ces deux fonctions de valeurs sont liées et il est facile de passer de l'une à l'autre en utilisant la relation suivante :

$$Q_\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \left[\mathcal{R}(s, a, s') + \gamma V^\pi(s') \right] \tag{5.5}$$

5.1.4 Estimation des fonctions de valeur

Les fonctions de valeur V_π et Q_π peuvent être estimées à partir de l'expérience acquise. Par exemple, si un agent suit une politique π et maintient une moyenne, pour chaque état rencontré, des retours effectifs obtenus à partir de cet état, alors cette moyenne va converger vers la valeur de cet état, $V_\pi(s)$, lorsque que le nombre de fois où cet état est rencontré tend vers l'infini. Si des moyennes séparées sont maintenues pour chaque action prise dans un état, alors ces moyennes vont également converger vers les valeurs de l'action, $Q_\pi(s, a)$. Les méthodes d'estimation de ce type sont appelées *méthodes de Monte Carlo* car elles calculent une moyenne sur des échantillons aléatoires des récompenses réelles.

Lorsque le PDM (en particulier $\mathcal{P}$ et $\mathcal{R}$) est intégralement connu, l'estimation des fonctions de valeur peut être vue comme un problème d'optimisation, et peut ainsi être résolue par une méthode de programmation dynamique. On parle alors de planification plus que d'apprentissage.

Une propriété fondamentale de ces fonctions de valeur est qu'elles satisfont certaines relations de récurrence. Pour toute politique π et tout état s, la condition de cohérence suivante est satisfaite entre la valeur de l'état s et de son

[3] $Q^*(s, a)$ pour la politique optimale

successeur éventuel :

$$
\begin{aligned}
V_\pi(s) &= E_\pi\left[R_t \mid s_t = s\right] \\
&= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} \mid s_t = s\right] \\
&= E_\pi\left[r_t \mid s_t = s\right] + \gamma E_\pi\left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+2} \mid s_t = s\right] \\
&= \sum_{a\in\mathcal{A}} \pi(s,\, a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,\, a,\, s')\mathcal{R}(s,\, a,\, s') \\
&\quad + \gamma \sum_{a\in\mathcal{A}} \pi(s,\, a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,\, a,\, s') E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k+1} \mid s_{t+1} = s',\, s_t = s\right] \\
&= \sum_{a\in\mathcal{A}} \pi(s,\, a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,\, a,\, s')\mathcal{R}(s,\, a,\, s') \\
&\quad + \gamma \sum_{a\in\mathcal{A}} \pi(s,\, a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,\, a,\, s') V_\pi(s') \\
V_\pi(s) &= \sum_{a\in\mathcal{A}} \pi(s,\, a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,\, a,\, s') \left(\mathcal{R}(s,\, a,\, s') + \gamma V_\pi(s')\right)
\end{aligned}
\tag{5.6}
$$

L'équation (5.6) ainsi obtenue est l'*équation de Bellman*. Si l'on écrit cette équation pour l'ensemble des états de $\mathcal{S}$, on obtient un système de $|\mathcal{S}|$ équations linéaires dont V_π est l'inconnue.

5.1.5 Fonctions de valeur optimales

L'équation de Bellman (5.6) s'applique également à la politique optimale et à sa fonction de valeur V^*, définie comme suit :

$$V^*(s) = \max_\pi V_\pi(s) = \max_{a\in\mathcal{A}} Q_\pi(s,\, a) \tag{5.7}$$

On peut donc en déduire l'*équation d'optimalité de Bellman* :

$$V^*(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}(s,\, a,\, s') \left(\mathcal{R}(s,\, a,\, s') + \gamma V^*(s')\right) \tag{5.8}$$

Cette politique π^*, et sa valeur $V^*(s)$ à chaque état $s \in \mathcal{S}$, peuvent être calculées en utilisant des algorithmes standards tels que l'algorithme d'itération de la politique ou de la valeur[4], dans le cas où $\mathcal{P}$ et $\mathcal{R}$ sont connus. Comme il est dit ci-dessus, ces algorithmes sont plutôt des algorithmes d'optimisation que réellement d'apprentissage.

Lorsque $\mathcal{P}$ et $\mathcal{R}$ ne sont pas connus, le problème est pleinement dans le domaine de l'apprentissage par renforcement. Il s'agit alors d'exploiter les conséquences des actions de l'agent sur son environnement. Il existe deux types d'approche : sans modèle et avec modèle[5]. Les méthodes avec modèle construisent un

[4] appelés en anglais *policy* et *value iteration*

[5] *model-based* et *model-free* en anglais.

modèle des fonctions $\mathcal{P}$ et $\mathcal{R}$ afin de retomber dans le cas où celles-ci sont naturellement connues. Les méthodes sans modèle apprennent en interagissant avec l'environnement au lieu d'interroger ces fonctions, sans construire de modèle de ces interactions.

L'algorithme Q-*Learning* est décrit dans la section suivante (section 5.1.6) et permet de calculer une approximation de Q^* lorsque $\mathcal{P}$ et $\mathcal{R}$ ne sont pas connus, indépendamment de la politique suivie.

5.1.6 L'algorithme Q-*Learning*

Les algorithmes mentionnés ci-dessus et basés sur l'équation de Bellman (5.6) ne peuvent être appliqués que dans le cas où l'environnement est connu, c'est-à-dire où les fonctions $\mathcal{R}$ et $\mathcal{P}$ sont connues.

Dans le cas où l'apprentissage par renforcement est appliqué à un système évoluant dans le monde réel, ce qui est notre cas, ces fonctions ne sont *pas* connues. Il faut donc remplacer cette connaissance par les interactions avec l'environnement réel pour apprendre la fonction qualité Q et utiliser un apprentissage sans modèle. C'est le principe de l'algorithme Q-*Learning* [Watkins, 1989] (algorithme 1).

Algorithme 1 : *L'algorithme* Q-*Learning*

Entrée : $\mathcal{P}$, $\mathcal{R}$
Sortie : π
Initialiser Q :
  $Q(s, a) \leftarrow 0,\ \forall (s, a) \in (\mathcal{S}, \mathcal{A})$;
Répéter
  $t \leftarrow 0$;
  Initialiser l'état initial s_0;
  Répéter
    Choisir une action $a = a_t = \pi(s_t)$ et l'exécuter;
    Observer l'état suivant $s' = s_{t+1}$ et la récompense $r = r_t$;
    Utiliser l'expérience $\langle s, s', a, r \rangle$, où $s = s_t$,
    pour mettre à jour Q :
      $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$[a];
    $t \leftarrow t + 1$;
  Jusque *la satisfaction du critère d'arrêt*[b] ;
Jusque ∞[c] ;

[a] Dans cette équation, α est le taux d'apprentissage et décroît au fur et à mesure que le comportement est bien appris.

[b] Ce critère d'arrêt peut être $s_t \in \mathcal{F}$, où $\mathcal{F}$ est l'ensemble des états finaux. Si un tel ensemble n'est pas défini pour l'application, ce critère peut être par exemple un nombre maximal d'itérations ou encore $|Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)| < \epsilon,\ \forall (s_t, a_t) \in (\mathcal{S}, \mathcal{A})$, où ϵ est un seuil prédéfini.

[c] Un pas de cette boucle est appelé *épisode*.

5.1.6.1 Exploration et exploitation

Dans l'algorithme Q-*Learning* (algorithme 1) ci-dessus, il est dit qu'à chaque état s, une action $a = \pi(s)$ est choisie en suivant la politique π courante. Le

choix de l'action par la politique doit assurer un équilibre entre exploration et exploitation. L'exploitation consiste à choisir la meilleure action pour l'état courant (l'action ayant la Q-*valeur* maximale), donc à exploiter la connaissance qu'a l'agent à l'instant présent de l'application. L'exploration consiste à choisir une action autre que la meilleure, *a priori* sous-optimale, afin de la tester, observer ses conséquences, et augmenter la connaissance de l'agent en améliorant sa Q-*table*. Il est logique de favoriser l'exploration au début de l'apprentissage, afin de rapidement couvrir une grande partie de l'espace d'états-actions, puis de la diminuer au profit de l'exploitation des connaissances acquises afin d'agir de manière à maximiser les récompenses obtenues.

Il existe plusieurs stratégies pour trouver cet équilibre entre exploration et exploitation, nous allons nous intéresser aux deux stratégies suivantes :

Gloutonne [6] Cette stratégie ne fait pas véritablement d'exploration car elle consiste à choisir toujours la meilleure action par rapport à la Q-*table* courante : $a = \arg\max_{a \in \mathcal{A}} Q(s, a)$.

ϵ-gloutonne Cette stratégie ajoute la partie exploration à la politique gloutonne, en choisissant au hasard à chaque étape si la politique va renvoyer l'action gloutonne (probabilité ϵ) ou bien une action aléatoire (probabilité $1 - \epsilon$) :

$$a = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s, a) & \text{avec une probabilité } \epsilon \\ \text{action tirée au hasard dans } \mathcal{A} & \text{avec une probabilité } 1 - \epsilon \end{cases}$$

Avec $\epsilon = 0$, la politique 0-gloutonne choisit toujours une action au hasard et ne fait aucune exploitation.

Il existe encore bien d'autres stratégies, notamment *softmax* et *boltzmann* qui choisissent une action en fonction de sa qualité relativement à la qualité des autres actions dans le même état. Pour de plus amples informations, le lecteur pourra se référer à [Sutton et Barto, 1998].

5.1.7 Traces d'éligibilité

Jusqu'à présent, nous avons supposé que la dernière récompense obtenue ne dépend que de la dernière transition effectuée. En réalité, cela peut être plus compliqué. L'exemple d'un robot mobile permet d'illustrer cette idée. Dans le cas d'un robot se déplaçant dans un environnement en évitant les obstacles, la collision d'un obstacle provoque un fort renforcement négatif. Dans l'état précédent la collision, le robot a, par exemple, une vitesse de $20\,m/s$ et se trouve à $5\,cm$ de l'obstacle. Le problème reste markovien car la collision ne dépend bien que de l'état précédent, mais lorsque le robot est dans cet état, il ne peut plus rien faire pour éviter la collision. Dans les algorithmes vus jusqu'ici, le renforcement négatif sera répercuté à l'état immédiatement précédent, il pourra être appris que cet état est très indésirable, mais le robot n'apprendra pas à éviter la collision. Il serait plus intéressant de propager ce renforcement aux états passés pour savoir plus en avance quel risque l'on court, mais se rappeler des états précédents sort le problème du cadre markovien.

Au lieu de se tourner vers le passé, certains algorithmes proposent, à chaque pas, d'évaluer le résultat d'un comportement en regardant vers le futur. Ainsi,

[6] *Greedy* en anglais.

lorsque le robot est à 50 *cm* de l'obstacle et qu'il estime ce qui se passera s'il continue sa trajectoire en gardant sa vitesse, il recevra la forte récompense négative et pourra encore éviter le choc. Ce raisonnement correspond à la vue théorique dite *en avant (forward)*.

Cette idée est mise en œuvre grâce aux *traces d'éligibilité*. Un état s, ou un couple état-action (s, a), est éligible s'il est, au moins en partie, responsable du retour immédiat. L'éligibilité d'un état, ou couple état-action, est mise à jour lorsque ce dernier est visité. Le dernier état s, ou couple (s, a), possède donc l'éligibilité la plus grande. Le coefficient d'éligibilité augmente sa valeur à chaque nouveau passage dans l'état s, ou couple (s, a), associé, puis décroît exponentiellement au cours des itérations suivantes, jusqu'à un nouveau passage dans cet état (ou couple état-action). Lors de la mise à jour de la valeur d'un état (ou d'un couple état-action), la différence temporelle[7] sera pondérée par l'éligibilité de cet état (ou couple). La propagation des valeurs est ainsi accélérée. On notera $e(s)$, ou $e(s, a)$, l'éligibilité d'un état ou d'un couple état-action. Les traces d'éligibilité correspondent à la vue mécanique dite *en arrière (backward)*.

Il peut être montré que les vues *en avant* et *en arrière* sont équivalentes [Sutton et Barto, 1998].

Les algorithmes classiques de l'apprentissage par renforcement (AR[8]) peuvent utiliser facilement les traces d'éligibilité. Nous allons expliciter ici l'extension aux traces d'éligibilité de l'algorithme Q-*Learning* (algorithme 1) : l'al-

[7] La quantité notée δ dans l'algorithme 2.

[8] Notation abrégée de « apprentissage par renforcement » qui pouura être employée dans la suite.

gorithme *Q(λ)* (algorithme 2).

Algorithme 2 : *L'algorithme Q(λ)*

Entrée : $\mathcal{P}$, $\mathcal{R}$
Sortie : π
Initialiser Q :
  $Q(s, a) \leftarrow 0$, $\forall (s, a) \in (\mathcal{S}, \mathcal{A})$;
Répéter
  $t \leftarrow 0$;
  $e(s, a) \leftarrow 0$, $\forall (s, a) \in (\mathcal{S}, \mathcal{A})$;
  Initialiser l'état initial s_0;
  Choisir l'action à émettre a_0;
  Répéter
    Exécuter l'action $a = a_t$;
    Observer l'état suivant $s' = s_{t+1}$ et la récompense $r = r_t$;
    Choisir l'action qui sera émise dans s' : $a' = a_{t+1}$;
    $a^* \leftarrow \arg\max_{a \in \mathcal{A}} Q(s', a)$;
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$;
    $e(s, a) \leftarrow e(s, a) + 1$;
    Pour $(s, a) \in (\mathcal{S}, \mathcal{A})$ **faire**
      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$;
      Mettre à jour l'éligibilité de (s, a) :
      si $a = a^*$ **alors**
        $e(s, a) \leftarrow \gamma\lambda e(s, a)$;
      sinon
        $e(s, a) \leftarrow 0$;
    [a];
    $t \leftarrow t + 1$;
  Jusque *la satisfaction du critère d'arrêt*[b] ;
Jusque ∞ ;

[a] Ceci est la méthode *Watkins* pour la mise à jour de l'éligibilité. Il existe également la méthode *naïve* et la méthode *Peng*, décrites dans [Sutton et Barto, 1998].

[b] Ce critère d'arrêt peut être $s_t \in \mathcal{F}$, où $\mathcal{F}$ est l'ensemble des états finaux. Si un tel ensemble n'est pas défini pour l'application, ce critère peut être par exemple un nombre maximal d'itérations ou encore $|Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)| < \epsilon$, $\forall (s_t, a_t) \in (\mathcal{S}, \mathcal{A})$, où ϵ est un seuil prédéfini.

5.1.8 Application à un environnement réel

Comme nous l'avons déjà mentionné dans la section 2.3, l'apprentissage doit être rapide, sans solliciter l'utilisateur trop fréquemment. Or les méthodes d'apprentissage sans modèles impliquent l'environnement réel, et donc l'utilisateur, à chaque pas. De plus, le nombre de pas nécessaire pour apprendre un comportement cohérent est très important, d'autant plus que l'environnement est complexe (et les espaces d'états et d'actions – vastes). Nous sommes alors dans l'obligation de revenir à des méthodes avec modèles sous peine d'obtenir un système inutilisable par les personnes. La section 5.2 explique comment cela est réalisé.

5.1.9 État de l'art

L'idée d'appliquer l'apprentissage par renforcement à des agents qui assistent l'utilisateur a été exploitée au Media Lab du MIT dès 1993 par [Kozierok et Maes, 1993, Maes et Kozierok, 1993]. Dans cet article, les auteurs présentent un agent d'interface qui aide l'utilisateur à gérer son agenda. Cet agent exploite des techniques d'apprentissage automatique pour s'adapter aux préférences de l'utilisateur en matière de règles de prise de rendez-vous. Une des particularités de cette approche est de fournir le contrôle à l'utilisateur sur la délégation graduelle à l'agent des tâches relatives à l'agenda. Ceci permet de construire progressivement une relation de confiance de l'utilisateur envers l'agent, ce que nous avons identifié dans la section 2.3 comme un des points clé de l'acceptation d'un agent intelligent par l'utilisateur. L'avantage d'utiliser l'apprentissage pour programmer un agent informatique est également que cela demande moins d'effort de la part des développeurs et des utilisateurs, et résulte en un agent mieux adapté aux préférences de chaque utilisateur particulier. L'apprentissage est basé sur deux techniques : l'apprentissage par analogie *(memory-based learning)* [Stanfill et Waltz, 1986] et l'apprentissage par renforcement. D'abord, l'agent observe l'utilisateur en enregistrant tout ce qu'il fait en tant que couples situation-action. Ensuite, dans une situation donnée, l'agent cherche la situation enregistrée la plus proche et peut ainsi prédire l'action à effectuer. L'apprentissage par renforcement intervient dans le cas où la prédiction était incorrecte. L'agent maintient un ensemble de poids associés aux initiateurs, participants et sujets des réunions. Dans le cas d'une mauvaise prévision, l'agent explique à l'utilisateur la raison de son choix (la situation similaire dans la mémoire de l'agent) et lui offre la possibilité d'expliquer en quoi le cas courant diffère de la situation précédemment rencontrée. Par exemple, l'utilisateur peut avoir accepté une réunion sur un sujet pour lequel il avait précédemment refusé une réunion parce que cette fois l'initiateur est son patron. Le poids du patron dans la liste des initiateurs sera alors augmenté d'une faible quantité. Cette approche est intéressante mais peut difficilement s'adapter à un système où les facteurs intervenant dans la prise de décision sont plus nombreux. Ici, l'on considère que la décision de l'utilisateur est basée sur trois facteurs (l'initiateur, les participants et le sujet). Dans un environnement ambiant, ces facteurs sont le contexte de l'utilisateur qui peut être complexe. Une situation peut être décrite en incluant l'heure et la date courantes, la luminosité de la pièce, l'activité de l'utilisateur, le lieu, le niveau sonore, la présence d'autres personnes et bien d'autres facteurs. Il n'est pas toujours évident pour l'utilisateur de décrire pourquoi exactement il a pris telle ou telle décision. Si les actions que l'agent peut effectuer dans l'environnement sont nombreuses, il se peut que l'agent pose trop souvent des questions à l'utilisateur, ce qui risque de provoquer le rejet du système.

Un autre exemple d'application de l'apprentissage par renforcement dans un système impliquant un utilisateur est un système de recommandations pour l'internet réalisé par [Hernandez *et al.*, 2004]. Dans ce travail, un agent de recommandations suggère des liens à un utilisateur parcourant un site Web (autres que les liens déjà présents dans la page courante). À chaque recommandation, l'agent reçoit d'office un renforcement négatif, reflétant le fait qu'une recommandation inadéquate nuit à l'utilisateur. Par contre, si l'utilisateur suit le lien suggéré, l'agent reçoit un renforcement positif car la suggestion a apparemment

aidé l'internaute. Le but de l'agent est de maximiser la somme de son renforcement reçu sur le long terme.

L'apprentissage par renforcement a également souvent été appliqué en robotique. En effet, programmer le comportement d'un robot mobile peut être très long et fastidieux étant donnée la complexité de l'environnement dans lequel évolue le robot. De plus, lorsque le concepteur spécifie les correspondances entre les capteurs bas niveau et les effecteurs, il utilise ses propres représentations, ce qui peut être épineux. L'idée de faire en sorte que le robot apprenne lui-même ces correspondances, en utilisant par conséquent ses propres représentations, semble judicieuse. Le programmeur se contenterait alors de spécifier au robot ce qu'il devrait faire, et le robot se charge d'apprendre comment le faire. Diverses techniques d'apprentissage ont été appliquées, par exemple l'inférence bayésienne [Lebeltel, 1999, Diard, 2003], nous allons nous concentrer sur l'apprentissage par renforcement.

[Smart et Kaelbling, 2002] ont appliqué une technique d'apprentissage par renforcement efficace aux robots mobiles. Dans cet article, les auteurs proposent de spécifier seulement la fonction de renforcement $\mathcal{R}(s, a)$, ce qui revient à spécifier une description haut niveau de la tâche que le robot doit effectuer. Les renforcements à donner correspondent à des événements dans le monde réel. Par exemple, pour un robot qui doit se déplacer en évitant les obstacles, on pourrait renvoyer un renforcement de 1 lorsque le but est atteint, et -1 lors d'une collision avec un obstacle. Les auteurs soulignent le fait qu'appliquer simplement un algorithme tel que le Q-*Learning* à un problème « réel » n'est pas si simple, des difficultés surviennent. Dans leur cas, il s'agit d'abord du fait que les algorithmes connus fonctionnent avec des espaces d'états discrets alors que l'espace d'états d'un robot est le mieux décrit par des vecteurs de réels. Ensuite, une fonction de renforcement telle que l'exemple donné ci-dessus est dite *éparse* (par opposition à *dense*). C'est-à-dire qu'elle ne renvoie des valeurs non nulles que rarement. Associée à un espace d'états très vaste (comme c'est le cas pour une application dans le monde réel), les performances du Q-Learning peuvent être catastrophiques. En effet, le système n'ayant aucune connaissance initialement, il n'a d'autre choix que d'essayer des actions aléatoires. La probabilité qu'il obtienne alors un renforcement non nul est très faible, et sans renforcement non nul, l'apprentissage ne peut « décoller ». Les auteurs argumentent qu'il faut alors introduire des *connaissances initiales* afin d'obtenir rapidement des renforcements non nuls. Nous allons rencontrer le même problème dans notre cas. En effet, nos renforcements sont fournis par l'utilisateur. Initialement, nous n'avons donc *que* des renforcements nuls. Notre espace d'états est également vaste. Nous allons donc également fournir des connaissances initiales à notre assistant (section 5.8). La solution proposée par [Smart et Kaelbling, 2002], mais qui ne s'applique pas à notre problème, est de fournir des trajectoires exemples au robot et de séparer l'apprentissage en deux phases. Lors de la première phase, c'est une politique fournie qui contrôle le robot, cette politique pouvant notamment être un humain contrôlant le robot avec une télécommande. Le système d'apprentissage se contente alors d'observer les états, actions et renforcements. Ceci lui permet de remplir la fonction de valeur avec quelques valeurs initiales. Les trajectoires exemples doivent faire passer le système d'apprentissage par renforcement par des états « intéressants », où les renforcements sont non nuls. Lorsque la fonction de valeurs est suffisamment complète, la politique apprise prend alors le contrôle du robot. Les résultats présentés dans cet article prouvent

l'efficacité de cette technique. La politique est apprise bien plus rapidement que le temps nécessaire à un programmeur humain pour spécifier un comportement. De plus, les trajectoires données en exemple sont effectuées d'une manière plus efficace par la politique apprise que par l'humain guidant le robot lors de la première phase.

Un autre exemple d'agent d'apprentissage par renforcement interagissant avec l'utilisateur est donné par [Walker, 2000]. Ici, l'utilisateur appelle par téléphone un système de gestion de ses e-mails appelé ELVIS. L'agent doit apprendre la manière optimale d'interagir vocalement avec l'utilisateur afin de lui fournir rapidement la bonne information. La ressemblance entre ce système et notre assistent réside dans la taille importante de l'espace des états. Un état d'ELVIS est défini par 13 variables discrètes, pouvant prendre différentes valeurs (entre deux et quatre valeurs possibles pour chaque variable). Les auteurs soutiennent que réduire l'espace d'états permet une amélioration significative des performances du système. Ils proposent alors de généraliser les états afin de les regrouper et donc réduire leur nombre. Pour cela, ils proposent d'ignorer certaines variables non pertinentes pour l'apprentissage. Ainsi, moins d'échantillons de dialogues sont nécessaires à l'apprentissage. Les états qui restent distingués sont seulement ceux pour lesquels différentes stratégies de dialogues sont explorées.

5.2 Apprentissage par renforcement indirect

L'apprentissage par renforcement en général est connu pour être lent. [Kaelbling, 2004] donne l'exemple du problème de l'ascenseur (trouver la trajectoire optimale) qui a nécessité 60000 heures de simulation, et celui du jeu Backgammon qui a appris sur plus de 1.5 millions de parties. Dans l'algorithme Q-*Learning*, ainsi que dans sa version étendue aux traces d'éligibilité, *Q(λ)* (algorithmes 1 et 2), un pas d'apprentissage est effectué lorsque l'on a émis une action dans un état, et observé l'état suivant ainsi que la récompense éventuelle (le cas où aucune récompense n'est reçue est traité comme le cas d'une récompense nulle). Ces derniers sont fournis par l'environnement. Or, ceci est peu adapté au cas où l'environnement est réel et où chaque action implique l'utilisateur. Les états et les actions sont de très haut niveau et le pas de temps entre deux changements d'état ou l'émission de deux actions peut être très important (de l'ordre de plusieurs minutes, voire dizaines de minutes). Dans un tel cadre, l'apprentissage est extrêmement lent. De surcroît, l'utilisateur ne doit pas être sollicité pour effectuer chaque pas d'apprentissage car il risquerait alors de rejeter rapidement le système (d'autant plus durant les phases d'exploration).

L'idée de l'apprentissage par renforcement indirect est d'ajouter aux expériences réelles des expériences virtuelles. Les interactions avec le monde réel ne suffisent pas pour apprendre la politique car elles sont trop rares. De plus, si l'environnement est complexe, alors l'espace d'états est vaste et certains états ne seront que très rarement visités. La politique risque donc d'être mauvaise pour ces états. L'apprentissage par renforcement indirect introduit un monde virtuel, un modèle du monde réel, dans lequel l'agent peut jouer autant d'expériences que nécessaire afin d'apprendre rapidement, sans impliquer l'utilisateur ni l'environnement réel. Autrement dit, le modèle du monde est utilisé pour générer des expériences imaginaires, telles que les a introduites Kenneth Craik en 1943. [Craik, 1943] développe la proposition que l'Homme utilise des modèles

mentaux, qu'il construit des « modèles réduits » de la réalité employés pour raisonner, anticiper les événements et servir de base aux explications.

L'apprentissage par renforcement indirect imite ce fonctionnement pour faire des essais par la pensée, de la simulation mentale. On applique les méthodes classiques d'apprentissage par renforcement sur ces expériences mentales exactement de la même façon que si elles avaient réellement eu lieu. On peut également utiliser le modèle du monde pour rejouer des expériences qui ont réellement eu lieu.

Ce modèle du monde doit représenter des connaissances générales sur l'environnement, des connaissances telles que « quelle est la pièce derrière cette porte » ou encore « comment se rendre à l'aéroport ». De plus, les connaissances doivent être représentées à différentes échelles. Le choix des variables d'état, et donc de la représentation, est fondamental. *Les trois choses importantes pour construire un système d'intelligence artificielle efficace sont la représentation, la représentation et la représentation* (Richard Sutton). Avec les bonnes représentations, l'apprentissage et la planification peuvent être rapides, sans elles, l'apprentissage peut ne jamais décoller.

La construction du modèle du monde est un processus continu et à long terme. C'est ce qui donne vie au système, ce qui lui permet de réagir aux changements du monde. La planification est un processus secondaire qui s'effectue en arrière plan.

Ce modèle du monde doit refléter la dynamique de l'environnement. Il est constitué d'un modèle de la fonction de transition $\mathcal{P}$ et d'un modèle de la fonction de récompense $\mathcal{R}$. Dans la plupart des cas, ces deux modèles ne peuvent pas être spécifiés (par exemple, si l'environnement est trop complexe ou si l'on ne connaît pas sa dynamique). Leur construction se fait donc par apprentissage en observant l'environnement, par exemple en appliquant une technique d'apprentissage supervisé.

5.2.1 Méthode DYNA d'apprentissage par renforcement indirect

La méthode DYNA a été introduite par Richard Sutton dans [Sutton, 1991]. Elle consiste en trois parties qui sont chacune répétées en permanence et sans arrêt :

1. Apprendre un modèle du monde ;
2. Utiliser ce modèle pour déterminer une bonne politique (étape de planification) ;
3. Déterminer une bonne politique sans utiliser le modèle (apprentissage par renforcement classique).

Ces trois parties sont asynchrones. La partie 1 s'exécute en parallèle et indépendamment des autres. Les parties 2 et 3 s'exécutent en alternance, comme illustré figure 5.3.

Cette méthode permet de revenir à un apprentissage avec modèle et de réutiliser les expériences passées pour accélérer le processus. En effet, le modèle du monde est fait pour *imiter* le monde réel. L'agent d'apprentissage exécute des actions dans le monde réel. Il dispose du modèle qui imite le monde et, à un certain moment, il échange les deux. Tout le reste de l'architecture ne change pas : les actions, états et récompenses sont de la même forme, l'algorithme

d'apprentissage est le même (par exemple TD*(0)*, *Sarsa* ou bien Q-*Learning*) et affecte la même fonction de valeur et donc la même politique. Tout se passe exactement comme précédemment, avec le monde réel. Cet échange est appelé DYNA *switch*. Lorsque l'agent interagit avec le monde réel, la phase d'apprentissage classique (phase 3) est en cours. La phase de planification (phase 2) a lieu lorsque le modèle a pris la place du monde réel. De cette façon, l'apprentissage et la planification sont « les mêmes », ils sont interchangeables. La figure 5.3 illustre ce principe. Cette méthode a largement été appliquée notamment dans [Sutton, 1990, Moore et Atkeson, 1993, Singh, 1992, Peng et Williams, 1993, Kuvayev et Sutton, 1996, Degris *et al.*, 2006b, Paduraru, 2007].

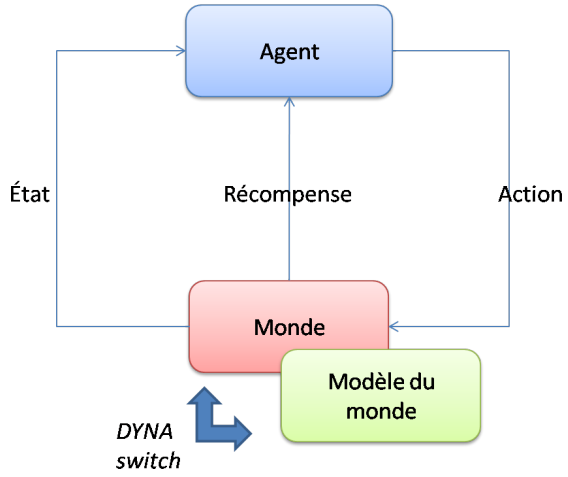

FIG. 5.3 – La méthode d'apprentissage par renforcement indirect DYNA : l'apprentissage s'effectue tour à tour avec le monde réel et le modèle du monde. L'échange des deux est appelé DYNA *switch*.

Par ailleurs, l'apprentissage et la planification sont incrémentaux, simultanés et asynchrones. L'un n'attend pas l'autre, tout est effectué en parallèle. Par exemple, pour un pas effectué dans le monde réel, l'agent effectue 10 ou 20 pas imaginaires en utilisant les modèles. L'apprentissage est donc 10 à 20 fois plus rapide que s'il était effectué seulement dans le monde réel.

Dans notre cas d'application, nous avons la particularité que l'utilisateur est directement impacté par le résultat de l'apprentissage. Si le comportement de l'agent change, l'utilisateur le remarque puisque les actions du système le concernent. Or, certains utilisateurs pourraient préférer que le comportement du système soit constant, et ne change qu'à des moments précis. [Greenberg, 2001] soutient que les systèmes sensibles au contexte complexes ont de fortes probabilités de faire des erreurs. Par conséquent, ces systèmes devraient être plutôt conservateurs dans leurs actions, ils devraient signaler ces actions très clairement, et devraient laisser l'utilisateur exécuter les actions risquées. [Bellotti et Edwards, 2001] ont un point de vue similaire. Il en est de même de [Kozierok et Maes, 1993, Maes et Kozierok, 1993] qui pensent que le comportement du système doit être suffisamment stable pour laisser le temps à l'utilisateur de le connaître et ainsi établir une relation de confiance. Pour que

le comportement de l'assistant soit prévisible pour l'utilisateur, il est possible de laisser ce dernier choisir entre différentes options : mettre le comportement à jour au fur et à mesure, à chaque DYNA *switch*, ou bien à intervalles de temps réguliers et prédéfinis, par exemple une fois par jour, on bien uniquement lorsque l'utilisateur le décide.

[Kaelbling, 2004] suggère un apprentissage avec modèle lorsque l'environnement est très complexe. L'auteur propose, pour apprendre à agir « dans la vraie vie », d'apprendre des modèles relationnels probabilistes concernant les effets des actions sur l'environnement.

5.2.2 État de l'art

Dans la section 5.1.9, nous avons vu que l'apprentissage par renforcement a plus de chances de succès lorsque la fonction de renforcement est dense. Même dans ce cas, le Q-Learning a besoin d'un grand nombre d'itérations pour converger. Ceci pose problème lors de l'application de l'apprentissage par renforcement à un problème réel (par opposition aux domaines d'application classiques, par exemple les jeux). Or, chaque itération correspond à une interaction avec l'environnement. Dans certains cas, par exemple lorsque l'utilisateur est dans la boucle, il n'est pas toujours possible de faire autant d'essais que nécessaire. [Bridle et McCreath, 2004] ont abordé ce problème. Les auteurs proposent un langage capable de capturer la structure complexe de l'environnement pour apprendre les transitions entre états. L'architecture DYNA de [Sutton, 1991], décrite section 5.2.1 est une approche générale pour traiter cette question.

Afin de mieux comprendre les possibilités offertes par l'apprentissage par renforcement, des études plus théoriques ont été réalisées. Par exemple, [Lin, 1990] réalise une étude comparative de deux algorithmes d'AR : le TD-*Learning* et le Q-*Learning*, et de leurs variantes utilisant un modèle de l'environnement. Cette étude s'effectue dans le contexte d'un environnement non déterministe et dynamique, ce qui simule la complexité du monde réel, mais ne l'atteint pas. Ici également, l'auteur explique que si la fonction de renforcement est éparse et/ou si le coût d'une erreur dans le monde réel est élevé, une bonne solution est d'apprendre un modèle du monde afin de pouvoir faire autant d'essais virtuels que nécessaire dans ce modèle. La manière d'utiliser le modèle de l'environnement en coopération avec le Q-Learning proposée par l'auteur est la suivante. Si la politique n'est pas sûre de l'action à choisir (s'il n'y a pas une action avec une valeur largement supérieure aux valeurs de toutes les autres actions possibles), alors on essaye toutes les actions pouvant être bonnes dans le modèle. On met alors à jour la fonction de valeur. On regarde ce qui va se passer au pas d'après en utilisant le modèle pour choisir l'action au pas courant. Cette méthode s'est avérée efficace, mais elle suppose d'avoir un bon modèle de l'environnement, ce qui n'est pas notre cas au départ.

5.3 Environnement partiellement observable et systèmes multi-agents

Certains problèmes réels peuvent être modélisés sous la forme d'un PDM, mais il est impossible de doter l'agent de capacités de perception suffisantes

pour observer l'état dans lequel il se trouve. L'agent ne peut alors pas observer directement le PDM qu'il utilise. Les *Processus décisionnels de Markov* ***partiellement observables*** (ou PDMPOs[9]) sont prévus pour modéliser ces problèmes [Aström, 1965, Smallwood et Sondik, 1973, Monahan, 1982, Kaelbling *et al.*, 1998].

Un PDMPO est un PDM complété d'un ensemble d'observations Ω et d'une fonction d'observation $\mathcal{O} : \mathcal{S} \times \Omega \to [0; 1]$ donnant la probabilité de l'observation o sachant l'état s par :

$$\mathcal{O}(s, o) = P(o_t = o \,|\, s_t = s) \tag{5.9}$$

L'agent ne connaît pas l'état réel s dans lequel se trouve l'environnement, mais seulement l'observation o. Il agit alors en fonction d'un état estimé b *(belief state)* défini par une distribution de probabilités sur $\mathcal{S}$.

Comme nous le verrons section 5.6.5, la résolution exacte d'un PDMPO est un problème PSPACE-*difficile* [Papadimitriou et Tsitsiklis, 1987], c'est-à-dire que la résolution est possible, mais en un temps illimité. Il existe toutefois différents algorithmes pour la résolution des PDMPO. L'algorithme *value iteration* a été adapté à ce cas et il en existe plusieurs variations, notamment l'algorithme RTDP-BEL de [Geffner et Bonet, 1998], l'algorithme FSVI *(Forward Search Value Iteration)* [Shani *et al.*, 2007], ou encore l'algorithme *Witness* [Kaelbling *et al.*, 1998].

Les approches usuelles sont plus « algorithmiques » et approximatives. Une famille de tels algorithmes sont les algorithmes HSVI *(Heuristic Search Value Iteration)* qui calculent la fonction de valeur optimale pour un sous-ensemble de l'ensemble d'états estimés (appelé *belief space)*, ce qui est considéré comme plus avantageux que d'avoir une fonction de valeur approximative pour tous les états estimés.

Par exemple, les algorithmes *grid-based* [Lovejoy, 1991] calculent la fonction de valeur pour un ensemble de points dans l'espace d'états estimés et font une interpolation pour déterminer la meilleure action à prendre pour d'autres états estimés que l'on rencontre et qui ne sont pas dans l'ensemble des points de la grille. De manière similaire, les approches *point-based* calculent une fonction de valeur sur un ensemble de points atteignables du *belief space* [Pineau *et al.*, 2003, Spaan et Vlassis, 2005, Smith et Simmons, 2005]. Des travaux plus récents [Hoey *et al.*, 2007] font appel à des techniques d'échantillonnage, de généralisation et d'exploitation de la structure du problème et ont permis d'étendre la résolution des PDMPO aux domaines avec un nombre d'états très important. La réduction de la dimensionnalité par des techniques d'ACP (analyse en composantes principales) ont également été étudiées [Roy et Gordon, 2003].

Les PDMPO trouvent un bon cas d'application dans des domaines où l'agent agit dans un environnement réel dont il n'a une perception que partielle, notamment la robotique ou encore les systèmes multi-agents. Dans ce dernier cas, chaque agent accède à son propre état et à celui de l'environnement, mais pas à l'état interne des autres agents. Seulement, ces autres agents exécutent des actions modifiant l'environnement. Ils sont considérés comme faisant partie du monde, par conséquent, chaque agent n'a qu'une observation partielle de l'environnement. De tels systèmes multi-agents en général ont été abordés notam-

[9] POMDP en anglais.

ment par [Littman, 1994]. Dans le cas où chaque agent est modélisé par un PDMPO, on parle de DEC-PDMPO (PDMPO décentralisé) et leur résolution a notamment été abordé par [Aras *et al.*, 2007] ou encore [Matignon *et al.*, 2007]. [Dibangoye *et al.*, 2009a, Dibangoye *et al.*, 2009b] ont également proposé plusieurs méthodes pour la résolution des DEC-PDMPO : itération de politique pour DEC-PDMPO avec renforcements dépréciés et [Dibangoye *et al.*, 2009c] ont proposé une méthode à base de points incrémentale d'élagage heuristique pour résoudre les DEC-PDMPOs à horizon fini. [Szer et Charpillet, 2006] proposent également une méthode de programmation dynamique à base de points pour la résolution des DEC-PDMPOs. [Hansen, 1998] a étudié des améliorations de l'algorithme d'itération de politique.

La possibilité d'application de ces formalismes à notre cas sera discutée dans la section 5.6.5.

5.4 Autres modèles basés sur les PDM

5.4.1 PDM factorisés (ou FMDPs)

Le modèle des PDM factorisés a été proposé par [Boutilier *et al.*, 1995] dans le but de résoudre de manière optimale des problèmes de planification dans un espace d'états très large. La plupart des méthodes existantes alors nécessitaient une énumération explicite de l'espace d'états qui croît exponentiellement avec le nombre de variables pertinentes au problème. Les autres méthodes étaient approximatives. Boutilier a proposé d'exploiter les régularités et les indépendances du domaine, c'est-à-dire la *structure* du problème, pour réduire l'espace d'états « effectif ».

Le problème est représenté sous la forme d'un PDM par la définition d'un ensemble de variables aléatoires, chacune prenant ses valeurs dans un ensemble fini. Un état est défini par une configuration particulière des valeurs de ces variables. Les transitions sont représentées par un ensemble de réseaux bayésiens dynamiques, un pour chaque action. Le réseau indique l'effet de l'action sur chaque variable. Cet effet peut être probabiliste. La factorisation réside dans le fait qu'on travaille sur des variables et non pas sur des instances comme c'est le cas dans les PDM classiques. La fonction de valeur (d'action) est également factorisée sous la forme d'un arbre. Les probabilités conditionnelles des nœuds post-action sont représentées par un arbre de décision. Cette représentation capture les indépendances entre affectations de variables. La fonction de récompense est une fonction de l'état du monde et est représentée par un diagramme d'influence atemporel.

[Boutilier *et al.*, 1995] proposent alors un algorithme d'itération de politique structuré (nommé SPI) pour apprendre la politique optimale dans un FMDP lorsque les fonctions de transition et de récompense sont connues.

Les travaux de Boutilier sont à la base des travaux de [Degris *et al.*, 2006a][10] qui proposent un apprentissage par renforcement indirect d'un FMDP. Ces travaux fournissent un cadre général, SDYNA, pour apprendre incrémentalement et sans connaissance a priori la structure d'un FMDP. La phase d'apprentissage structuré des arbres de décision correspondant aux fonctions de transition et de récompense est effectué par l'algorithme SPITI proposé par [Degris *et al.*, 2006a].

[10] voir aussi [Degris *et al.*, 2006b, Degris, 2007]

5.4.2 PDM relationnels (ou RMDPs)

Tandis que les FMDPs permettent d'exploiter les régularités et indépendances du problème, les RMDPs ajoutent à cela la prise en compte des relations entre éléments de l'environnement.

Cette idée a été introduite par [Džeroski *et al.*, 1998][11] avec l'*apprentissage par renforcement relationnel*. Cette approche inclut une représentation structurée des états, ce qui permet de décrire des mondes « infinis ». Un état est représenté par une liste de *faits* relationnels, par exemple, dans un monde de blocs, un état pourrait être $sur(a, b)$ signifiant que le bloc a est *sur* le bloc b. Contrairement à l'apprentissage par renforcement classique, l'AR relationnel permet de s'abstraire des spécificités telles que les instances particulières des blocs et d'adapter la politique apprise pour un but à un autre but[12]. L'adaptation est également possible si le monde change, par exemple si un bloc est ajouté. La Q-fonction est représentée sous la forme d'un arbre de régression logique appelé Q-*tree*. Dans cette méthode, l'agent ne connaît pas la dynamique de l'environnement (la fonction de transition), il effectue un apprentissage sans modèle, uniquement à partir d'interactions.

De manière similaire, [Guestrin *et al.*, 2003] ont introduit les PDM relationnels (ou RMDPs). Les auteurs vont plus loin que Džeroski en modélisant un PDM *patron* pour un domaine. Un PDM spécifique peut ensuite être obtenu pour chaque environnement particulier du domaine en instanciant le PDM générique. Le domaine est défini sous la forme d'un schéma, précisant un ensemble de *classes* d'objets. Chaque classe vient avec un ensemble de *variables d'état* décrivant l'état d'un objet de la classe. Chaque variable d'état prend ses valeurs dans un domaine défini. En plus de cela, le schéma définit un ensemble de *liens* entre classes. Une instance du schéma est définie par un monde spécifiant les objets de chaque classe présents dans ce monde.

Les transitions et récompenses d'un RMDP sont également définis au niveau du schéma. Chaque classe possède une variable d'état particulière représentant l'action exécutée par l'objet courant. Comme pour les autres variables, l'action possède un domaine de définition pour chaque classe, traduisant les capacités de chaque classe d'objets. De plus, pour chaque classe, on définit un modèle de transition représenté par une distribution de probabilités sur l'état suivant d'un objet de la classe étant donné l'état courant de cet objet, l'action qu'il a exécutée et les états et actions de tous les objets auxquels il est lié.

Un objet pouvant être lié à un nombre illimité d'autres objets du monde, il est nécessaire de fournir une représentation compacte de la fonction de transition. Ceci est obtenu par *agrégation*, en particulier par recensement du nombre d'objets liés.

Les récompenses sont définies selon l'état de l'objet et son action. Chaque objet contribue au renforcement global.

Étant donné un monde, le RMDP abstrait permet d'obtenir de façon unique un FMDP concret. L'utilisation d'un PDM factorisé (décrit section 5.4.1) permet d'exploiter la structure du monde et ainsi d'être efficace dans les mondes très larges. L'ensemble des variables aléatoires du FMDP est constitué de toutes les variables d'état de tous les objets du monde. La fonction de renforcement est dé-

[11] Voir aussi [Džeroski *et al.*, 2001].

[12] Par exemple, adapter la politique apprise pour le but $sur(a, b)$ au but $sur(c, d)$ sans réapprentissage.

finie comme la somme des renforcements renvoyés par tous les objets du monde. Chaque objet est capable d'exécuter une action parmi celles du domaine de sa classe. À chaque pas de temps, un objet est choisi. Cet objet agit en exécutant une de ses actions.

Le RMDP peut être résolu de manière approximative. On suppose que la fonction de valeur est bien approximée par la somme des sous-fonctions locales des objets individuels du monde. Toutefois, cette approche ne permet pas de déduire une solution pour un monde à partir de la solution pour un autre monde. Afin d'atteindre cette généralisation, les auteurs supposent que tous les objets de la même classe se comportent de façon similaire dans différents mondes : ils partagent le modèle de transition et la fonction de renforcement. Enfin, une sous-fonction de valeur locale est définie pour chaque classe. Les auteurs obtiennent ainsi une formule générique permettant d'obtenir la fonction de valeur d'un monde en sommant les sous-fonctions locales de chaque objet du monde.

Une fois les sous-fonctions de valeur locales obtenues pour chaque classe, il devient possible de passer d'un monde à l'autre sans replannification.

Toutefois, la structure du domaine (les fonctions de transition et de récompense) est supposée connue.

[Kersting *et al.*, 2004] ont conçu un algorithme relationnel d'itération de valeur, appelé REBEL, qui résout un RMDP en travaillant sur des états et actions abstraits. Un état abstrait représente un ensemble d'états en utilisant des variables. Ces variables peuvent être remplacées par des constantes pour retrouver les états concrets faisant partie de l'état abstrait. Dans le monde de blocs, un état abstrait est par exemple $on(X, Y)$ représentant tous les états dans lesquels un bloc est sur un autre.

Sur cette base, [Walker *et al.*, 2008] ont proposé un algorithme (nommé AMBIL) qui construit un modèle relationnel du monde en parallèle de l'apprentissage de la politique.

5.5 Notre approche

Nous voulons rendre des services à l'utilisateur dans notre environnement intelligent, en fonction de son contexte et de ses préférences. Comme nous l'avons détaillé dans la section 2.4, nous devons tenir compte de plusieurs contraintes dans la réalisation de notre système. Pour que notre assistant puisse agir, il lui faut un modèle de contexte définissant les états dans lesquels l'environnement peut se trouver et les actions associées à chaque état. Ce modèle est trop complexe pour être fourni entièrement par le développeur ou l'utilisateur. De plus, ce modèle repose sur l'environnement et les préférences de l'utilisateur qui sont tous deux dynamiques et vont évoluer au cours de la vie de l'assistant, ce qui rejoint la vision interactionniste du contexte de [Dourish, 2004] évoquée section 2.4. Le modèle de contexte va donc, lui aussi, évoluer. Nous avons établi au début de ce chapitre (page 92) que l'apprentissage par renforcement est une approche adaptée pour l'obtenir. En outre, il permet à l'utilisateur d'entraîner le système de manière très simple, en exprimant uniquement sa satisfaction (positive ou négative) envers les actions de l'assistant.

D'après la section 2.4, nous devons concevoir notre système de sorte que l'utilisateur puisse le comprendre. Ceci sera rendu vrai par la définition de notre agent d'apprentissage par renforcement décrit ci-dessous. De plus, nous avons

d'autres contraintes dues au fait que l'utilisateur est impliqué dans la boucle de l'apprentissage. Celui-ci doit donc être rapide et ne doit pas nécessiter une grande quantité d'exemples. Nous pouvons contourner ce problème en appliquant une architecture du type DYNA à notre problème. Enfin, l'apprentissage ne doit pas démarrer à zéro car le comportement initial ne doit pas être absurde. Pour cela, nous allons fournir un modèle du monde initial, par défaut et partiel.

Dans le but d'appliquer le modèle DYNA, nous pouvons distinguer deux parties à gérer : les interactions en direct avec l'utilisateur dans le monde réel, et l'apprentissage non interactif[13]. Nous allons commencer par donner une vue d'ensemble de notre système (section 5.5.1). Puis, nous détaillerons les différentes parties en commençant par expliciter la manière dont nous gérons les interactions réelles (section 5.6), c'est-à-dire nos espaces d'états et d'actions et la prise en compte des récompenses utilisateur. Ensuite, nous décrirons notre modèle de l'environnement (section 5.7), c'est-à-dire la fonction de transition et la fonction de récompense et nos algorithmes d'apprentissage non interactif (sections 5.9 et 5.10).

5.5.1 Vue d'ensemble

Le système global est constitué d'une partie qui se déroule en ligne et d'une partie qui est lancée régulièrement hors ligne, en tâche de fond.

Le fonctionnement en ligne est décrit dans la section 5.6.5. Pour résumer, il s'agit pour l'assistant de recevoir un événement d'un de ses capteurs et de l'interpréter comme changement d'état. Ainsi, une action est choisie par la politique de l'agent d'apprentissage par renforcement afin de réagir à cet événement (ou plutôt à ce changement d'état). L'assistant utilise ses modules effecteurs afin d'exécuter l'action dans l'environnement réel. L'utilisateur donne éventuellement un renforcement. Les transitions et renforcements sont sauvegardés dans la base de données en vue de l'apprentissage supervisé du modèle du monde.

La partie non interactive est décrite dans la section 5.10 et consiste à jouer des épisodes de Q-Learning en utilisant le modèle du monde plutôt que l'environnement réel.

Ce modèle est appris de manière supervisée (section 5.9), également en tâche de fond. Il est divisé en deux parties, le modèle de transition, décrit section 5.7.1 et le modèle de renforcement, décrit section 5.7.2. Étant donné que l'apprentissage du comportement s'effectue en utilisant le modèle qui est lui-même appris en parallèle, le comportement à un instant t de la vie de l'assistant dépend de la connaissance que ce dernier a du monde à ce moment t. Cette connaissance peut être incomplète car l'apprentissage supervisé n'a pas encore rencontré suffisamment d'exemples. Pour cette raison, le pas du Q-Learning ne s'effectue pas sur une partie du monde inconnue du modèle.

Le modèle est appris pendant toute la durée de vie de l'assistant afin de prendre en compte les changements de l'environnement et des préférences utilisateur. Ces modifications sont intégrées au modèle progressivement afin de le garder stable.

[13] « Hors ligne ».

L'algorithme 3 résume le fonctionnement global de l'assistant.

Algorithme 3 : L'algorithme global de l'assistant.

Entrée : Modèles de transition et de récompense initiaux.
Sortie : Le modèle de contexte de l'utilisateur.
Exécuter un épisode (algorithme 7);
$i = 0$;
répéter
  Recevoir le nouvel état s_i;
  Sauvegarder l'exemple suivant dans la base de données : $\{s_{i-1}, a_{i-1}, s_i\}$;
  Choisir une action en utilisant la politique courante $a_i = \pi(s_i)$ et retourner l'action à l'assistant afin qu'il l'exécute dans l'environnement réel;
  Afficher s_i et a_i pour l'utilisateur;
  Si l'utilisateur donne une récompense, alors la sauvegarder dans la base de données : $\{s_i, a_i, r_i\}$;
  si *i est un multiple de n,*[a] **alors**
    Exécuter l'apprentissage supervisé du modèle de transition (algorithme 5);
    Exécuter l'apprentissage supervisé du modèle de récompense (algorithme 6);
  $i = i + 1$;
jusqu'à ∞ ;
En parallèle, à des intervalles de temps réguliers, exécuter un épisode (algorithme 7);

[a] n est le nombre de pas entre deux mises à jour du modèle du monde. Il est possible de choisir n petit au début de la vie de l'assistant car il a « beaucoup de choses à apprendre ». Au fur et à mesure il est possible d'éloigner quelque peu les mises à jour car l'environnement n'évolue pas très rapidement (mais nous n'allons jamais les arrêter totalement).

5.6 Interactions avec l'environnement

5.6.1 Définition d'un état

La modélisation de notre système sous forme d'un PDM (processus décisionnel de Markov) nécessite la définition d'un espace d'états $\mathcal{S}$. Nous devons trouver un moyen de représenter tous les états de notre environnement tout en respectant une contrainte majeure : un état doit être compréhensible par l'utilisateur, comme il a été établi section 2.4. Ceci nous permettra de fournir des explications à l'utilisateur quant aux actions du système et ainsi gagner la confiance de l'utilisateur.

Nous avons choisi de modéliser un état par un ensemble de prédicats. Chaque prédicat concerne une partie de l'environnement qui est intéressante pour le système. Vu la complexité de l'environnement, des prédicats d'ordre zéro ne sont pas suffisants. Nous utilisons des prédicats du premier ordre afin d'intégrer des valeurs libres dans la définition d'un état. Un prédicat admet des arguments pouvant prendre des valeurs quelconques. Par exemple, l'un des pré-

dicats définissant un état est `alarm(title, hour, minute)` correspondant à un rappel de l'agenda de l'utilisateur. Lorsqu'il n'y a pas de rappel, le prédicat `alarm` reste présent dans l'état courant, mais ses arguments ont des valeurs spéciales (`<null>`) signifiant l'absence de valeur. Lorsque l'état courant contient `alarm(title=<null>, hour=<null>, minute=<null>)`, ceci signifie qu'il n'y a pas de rappel à cet instant. En résumé, un état est défini par une liste de prédicats, chacun étant présent exactement une fois et dont les valeurs des arguments varient. Ces prédicats sont décrits dans le tableau 5.1.

Le choix des prédicats n'est pas le choix idéal du point de vue de l'apprentissage par renforcement car il complique le système. Toutefois, il se justifie étant donnée notre contrainte majeure, la compréhensibilité. [Ranganathan *et al.*, 2004] ont également adopté la représentation par prédicats des états dans leur système ambiant *Gaia*. Dans cette application, le développeur peut être amené à manipuler directement les prédicats, ce qui est possible grâce à leur caractère lisible. Les auteurs apportent les arguments supplémentaires que les prédicats permettent de faire facilement du raisonnement à base de règles et de décrire le contexte indépendamment de tout langage de programmation, système d'exploitation ou intergiciel. En effet, avoir une représentation directe et non pas un certain codage permet à différents composants d'accéder à l'information du contexte.

Les prédicats définissant notre espace d'états ont été choisis en fonction des capteurs à notre disposition dans l'environnement (il s'agit des modules décrits section 4.5.1). Si nous installions de nouveaux modules, par exemple un détecteur d'intensité lumineuse et un module pour l'exploiter, nous ajouterions un prédicat `lightIntensity(value, office)`, fournissant l'information de la valeur de l'intensité lumineuse (argument `value`) dans un bureau particulier (argument `office`) de l'environnement. De même, si nous ne disposions pas de fiches Bluetooth, nous utiliserions un autre moyen pour détecter les événements d'entrée et de sortie d'utilisateurs, par exemple un lecteur de puces RFiD ou bien une caméra vidéo enrichie d'un logiciel de détection de personnes. Les arguments des prédicats « `entrance` » et « `exit` » seraient alors différents. Idéalement, on peut envisager de sélectionner automatiquement les prédicats à utiliser lors de l'installation du système dans un nouvel environnement. Ce point est abordé en perspective 8.2.

Voici un exemple d'état :

```
alarm(minute=<null>, title=<null>, hour=<null>) ;
xActivity(isActive=<null>, machine=<null>) ;
inOffice(office=<null>, user=<null>) ;
absent(user=<null>) ;
hasUnreadMail(from=<null>, to=<null>, body=<null>, subject=<null>) ;
entrance(isAlone=<null>, friendlyName=<null>, btAddress=<null>) ;
exit(isAlone=false, friendlyName=Sonia, btAddress=00:12:47:C9:F2:AC) ;
task(taskName=<null>) ;
user(login=zaidenberg) ;
userOffice(office=E214, login=zaidenberg) ;
userMachine(login=zaidenberg, machine=hyperion) ;
computerState(machine=hyperion, isScreenLocked=<null>,
     isMusicPaused=<null>) ;
```

Dans l'exemple ci-dessus, l'utilisateur principal est `zaidenberg` et un dispositif portable Bluetooth dont l'adresse est `00:12:47:C9:F2:AC` vient de quitter

Prédicat	Arguments	Description
alarm	title, hour, minute	Le dernier rappel émis par l'agenda de l'utilisateur ; Lorsqu'il n'y a pas de rappel, les arguments ont des valeurs nulles.
xActivity	machine, isActive	Indique s'il y a une activité clavier/souris sur un ordinateur donné.
inOffice	user, office	Indique, s'il est connu, le bureau dans lequel se trouve l'utilisateur, null sinon.
absent	user	Déclare que l'utilisateur n'est pas dans son bureau. Il pourrait être dans un autre bureau ou bien le système ne sait pas où il se trouve.
hasUnreadMail	from, to, subject, body	Le dernier mail reçu par l'utilisateur.
entrance	isAlone, friendlyName, btAddress	Indique que quelqu'un vient d'entrer dans le bureau de l'utilisateur. Cette personne est identifiée grâce à son dispositif Bluetooth. L'argument *isAlone* indique si l'utilisateur était seul dans son bureau avant cet événement.
exit	isAlone, friendlyName, btAddress	De la même manière que précédemment, indique que quelqu'un vient de quitter le bureau de l'utilisateur.
task	taskName	La tâche sur laquelle l'utilisateur est en train de travailler.
user	login	Ce prédicat n'est pas sensé être modifié, il désigne l'utilisateur principal de l'assistant.
userOffice	office, login	De même, ce prédicat identifie le bureau de l'utilisateur principal.
userMachine	machine, login	De même, ce prédicat identifie l'ordinateur de l'utilisateur principal.
computerState	machine, isScreenLocked, isMusicPaused	Décrit l'état de l'ordinateur de l'utilisateur en indiquant si son écran est verrouillé ou non et si la musique est en pause ou non.

TAB. 5.1 – Les prédicats qui définissent un état du PDM modélisant l'environnement.

la pièce. La base de données, décrite dans la section 4.4, nous informe que ce dispositif Bluetooth appartient à l'utilisateur `zaidenberg`, nous en déduirons donc que l'utilisateur est absent du bureau : `absent(`*`user`*`=zaidenberg)`. Le prédicat `task` correspond à la tâche courante de l'utilisateur. Il s'agit d'une tâche « de haut niveau », telle que « préparer une réunion » ou bien « rédiger une thèse ». L'utilisateur définit ses propres tâches et sélectionne la tâche courante grâce à une icône toujours présente dans sa zone de notification. Cette interface a été

décrite section 4.4.

De plus, chaque prédicat possède une estampille[14] qui compte le nombre de pas (ou changements d'état) depuis le dernier changement des valeurs de ce prédicat. Entre autres, cette estampille sert à maintenir l'intégrité des états. Ce point est développé section 5.6.2.

Cette définition d'un état ne prévoit aucune restriction sur les valeurs des arguments, ce qui génère un espace d'états d'une taille illimitée. Même en se limitant aux valeurs plausibles, nous obtenons un espace immense.

5.6.1.1 Domaine relationnel

On peut voir un parallèle fort entre la définition des états donnée ci-dessus et les PDM relationnels présentés section 5.4.2.

Les RMDPs définissent des *classes* d'objets et fournissent une solution générique au niveau des classes. Il est ensuite facile de passer d'un environnement à l'autre, avec différentes configurations des instances des classes. Il n'est pas nécessaire de refaire un apprentissage.

Nous pouvons voir nos prédicats comme des classes. Les arguments des prédicats deviennent alors des variables d'état des classes. Il existe des liens entre éléments de notre environnement, mais ces liens ne sont pas explicitement représentés dans nos états. La notion de liens des RMDPs pourrait nous être utile. Par contre, nous devrions adapter ce qui a été fait par [Guestrin *et al.*, 2003] pour travailler avec des domaines de définition des variables qui ne sont pas finis, ou, du moins, que nous ne connaissons pas à l'avance (qui sont complétés au long de la vie du système).

Tel que notre système est conçu, il n'y a, dans un monde, qu'une seule instance de chaque classe. Comme il sera développé section 5.6.2, nous réagissons à un événement à la fois, puis nous l'oublions. Par exemple, même si plusieurs rappels sont émis à la fois, nous les traitons séquentiellement. Nous n'aurions pas plusieurs instances de la classe `alarm`, mais une seule dont les valeurs des variables d'état changent.

L'intérêt principal d'appliquer les RMDPs est de pouvoir changer de monde sans réapprentissage, un monde étant caractérisé par la configuration des instances des classes. Nous n'avons qu'une seule instance de chaque classe, par conséquent, nous n'exploiterions pas cet avantage.

Dans les perspectives (section 8.2), nous aborderons l'idée d'introduire la notion de *lieu* dans notre assistant, permettant à l'utilisateur de transporter ses préférences d'un lieu à l'autre (le bureau, la maison, la voiture, etc.). On pourrait voir ces lieux comme des mondes, et transposer le résultat de l'apprentissage d'un monde à l'autre permettrait de conserver le comportement appris pendant chaque passage dans un monde. Le problème qui se pose alors est celui du domaine de définition des variables. Comme il est dit plus haut, nous n'avons pas un domaine fixe, mais nous devons le compléter au fur et à mesure. Le domaine de définition sera certainement différent d'un monde à l'autre car les valeurs des variables d'état concerneront des parties distinctes de la vie de l'utilisateur. Le comportement appris n'apparaît plus comme étant toujours transposable d'un monde à l'autre dans ce cas.

[14] *timestamp* en anglais

5.6.1.2 Réduction de l'espace d'états

Il n'est pas envisageable de travailler avec une Q-table d'une taille aussi importante que celle imposée par notre définition de l'espace d'états. De plus, garder chaque état spécifique dans la Q-table réduit les performances du système. En effet, il n'est probablement pas pertinent de garder dans la Q-table deux entrées distinctes pour l'état où M. Dupond entre dans le bureau de l'utilisateur et l'état où c'est M. Martin qui entre. L'action à exécuter a de fortes chances d'être la même pour ces deux états. Ainsi, il serait judicieux de généraliser les deux états en un seul où *quelqu'un* entre dans le bureau. Lorsque l'on observe pour la première fois un visiteur, le système apprend une estimation du comportement à avoir dans le cas où n'importe qui entre dans le bureau. Ceci est plus efficace car il ne faut pas attendre de voir chaque personne pour agir de manière cohérente.

Pour réaliser ceci, nous généralisons les états selon les valeurs des arguments. La Q-table contient des états génériques et a, par conséquent, une taille largement réduite. Cette généralisation se fait en oubliant les valeurs exactes des arguments perçus et en ne distinguant que les valeurs nulles des valeurs non nulles. Ainsi, la Q-table contient des états dont les valeurs des arguments ont toutes été remplacées par l'un des deux caractères *joker* : `<+>` et `<*>`. Le symbole `<+>` remplace toutes les valeurs non nulles, alors que le symbole `<*>` remplace les valeurs nulles `<null>`. Cette technique d'agrégation d'états est également utilisée par [Walker, 2000, Kaelbling, 2004].

Lorsque l'assistant a besoin d'agir, la politique doit rechercher l'état courant dans la Q-table et renvoyer une des actions correspondantes. Or, l'état courant contient des valeurs (telles que `absent(`*`user`*`=zaidenberg)`) alors que la Q-table ne contient que des caractères joker. Par conséquent, l'état courant est d'abord généralisé et c'est cet état généralisé qui est recherché dans la Q-table. La généralisation s'effectue en suivant trois règles simples. Premièrement, les valeurs booléennes ne sont pas généralisées. En effet, il est difficile de considérer que le comportement doit être le même si un argument vaut « vrai » *et* s'il vaut « faux ». Deuxièmement, les valeurs nulles (`<null>`) sont remplacées par `<*>`. Troisièmement, toutes les autres valeurs sont remplacées par `<+>`. Sur l'exemple ci-dessus nous obtenons : `absent(`*`user`*`=<+>)`. Nous obtenons donc une Q-table dont un extrait est montré figure 5.5.

Cette idée est similaire à celle des systèmes de *classeurs* décrits par [Sigaud et Gérard, 2005]. Dans ces systèmes, ce ne sont pas les couples état-action qui sont associés dans la Q-table, mais les couples condition-action, une condition étant un ensemble de contraintes pouvant être vérifiées par plusieurs états, ce qui permet la généralisation. Un tel système n'a pas accès à l'état exact de l'environnement, mais à une situation composée d'attributs perçus.

D'autres méthodes sont appliquées dans la littérature pour réduire l'espace d'états. Notamment, il est possible d'exploiter la structure du problème afin de fournir une représentation plus compacte, tel qu'il est fait dans les PDM factorisés présentés section 5.4.1.

L'état ne serait alors plus défini par des prédicats mais par les attributs des prédicats directement, qui seraient considérés comme des variables aléatoires. Afin de garder l'aspect lisibilité par un humain des états, il serait possible de garder la notion de prédicat implicitement, sans la faire intervenir dans la définition formelle du système. Les fonctions de transition et de récompense sont repré-

sentées sous la forme de réseaux bayésiens dynamiques et la fonction de valeur d'action est représentée sous forme structurée par un arbre. [Degris *et al.*, 2006a] proposent un algorithme d'apprentissage indirect d'un FMDP.

5.6.1.3 Division d'états génériques

La généralisation de toutes les variables d'un état parait excessive. Dans certains cas, il peut être pertinent de distinguer les valeurs concrètes des arguments pour choisir l'action. Par exemple, l'utilisateur pourrait vouloir être averti d'un e-mail si l'expéditeur est son supérieur hiérarchique (`from=directeur@entreprise.com`), mais pas s'il s'agit d'un bulletin (`from=newsletter@lemonde.fr`). Il faut prévoir un mécanisme de division d'états.

Il n'est pas possible de savoir à l'avance quelles sont les valeurs qui devront être explicites dans la Q-table, mais nous pouvons effectuer un post-traîtement pour diviser certains états génériques en plusieurs états contenant des valeurs concrètes. La piste à explorer pour réaliser ceci est d'analyser l'historique des renforcements donnés par l'utilisateur et de chercher les renforcements contradictoires pour un même état générique et une action. Sur l'exemple de l'e-mail, l'état générique contient simplement le joker `from=<+>`. Dans l'historique nous pourrions retrouver tous les états non génériques qui se généralisent en cet état. Si, pour une même action, tous les renforcements sont du même ordre de grandeur, alors l'état générique est justifié. Si, au contraire, on observe une grande variation des renforcements, il faut alors chercher des correspondances entre valeurs et renforcements. Si l'on trouve une certaine cohérence, on peut alors diviser l'état générique de la Q-table en plusieurs états, dont certains spécifiques. Par exemple, un état serait spécifique au directeur, et un état générique serait appliqué à tous les autres expéditeurs. Lors de la recherche dans la Q-table, il faudrait alors en premier rechercher parmi les états spécifiques, avant de considérer les états génériques.

Dans les systèmes de classeurs évoqués ci-dessus, il arrive également qu'une condition soit trop générale. Un mécanisme de spécialisation est alors prévu. Le système de classeurs à anticipation MACS introduit par [Gérard *et al.*, 2005] effectue un apprentissage par renforcement indirect avec généralisation. Chaque attribut générique des conditions des classeurs est associé à une estimation i_s de l'amélioration apportée par la spécialisation de cet attribut. Cette estimation est obtenue à partir de l'historique des situations ayant mené à une bonne ou une mauvaise anticipation. Le système vérifie l'égalité de chaque attribut de la situation avec la valeur anticipée. Selon les cas, le coefficient i_s décroît ou croît. Lorsqu'un classeur « oscille », c'est-à-dire qu'il anticipe parfois correctement et parfois incorrectement, un processus de spécialisation est appliqué en se basant sur les coefficients i_s. Le classeur est divisé en plusieurs classeurs plus spécialisés.

5.6.2 Intégrité des états

Les changements d'état sont orchestrés par l'assistant personnel, en fonction des informations qu'il reçoit des autres modules de l'environnement. Avant de soumettre un changement d'état au processus d'apprentissage, il est nécessaire de vérifier sa cohérence. Par exemple, lorsque l'utilisateur entre dans le bureau, il ne faut pas seulement remplir le prédicat `inOffice`, mais également vider le

prédicat `absent` car les deux ne peuvent pas avoir des valeurs non nulles dans un même état. Ceci est effectué par l'agent d'AR en se basant plutôt sur la valeur de l'estampille des prédicats afin d'être indépendant de l'assistant et de pouvoir vérifier la cohérence d'un état à tout moment. Les valeurs du prédicat dont l'estampille a la plus haute valeur (donc le prédicat le moins récemment mis à jour) sont effacées. En effet, ce prédicat est alors considéré obsolète. Un autre exemple est le prédicat `alarm` qui ne garde sa valeur que pour un pas. Lorsqu'un rappel est émis, une action est immédiatement choisie et exécutée, puis le rappel est effacé. Cette action peut être de ne rien faire, mais une décision est prise immédiatement.

Ce mécanisme est modélisé sous forme d'un ensemble de règles sur les valeurs d'arguments de prédicats ou bien sur les estampilles. Ces règles sont enregistrées dans la base de données (décrite section 4.4), dans les tables de la figure 5.4. Une règle est l'association d'une ou plusieurs parties gauches (les conditions à remplir pour déclencher la règle) et d'une ou plusieurs parties droites (les actions à exécuter si toutes les conditions sont remplies). Les actions possibles sont (pour un argument) : effacer la valeur (la remplacer par `<null>`), mettre une valeur donnée ou bien mettre la valeur d'un autre argument (d'un autre prédicat). Les parties gauches et droites font référence aux prédicats et à leurs arguments, également enregistrés dans la base.

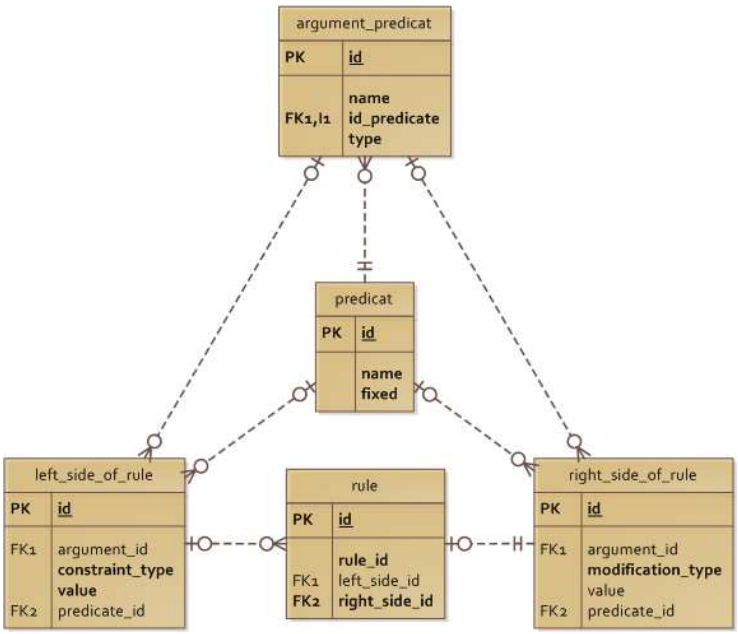

FIG. 5.4 – Une partie du schéma *services* de la base de données.

5.6.3 Définition d'une action

Notre ensemble d'actions est formé de toutes les actions que nous pouvons exécuter dans l'environnement. Il serait facile de rajouter des actions si nous développons d'autres modules effecteurs. Ajouter une action en cours de route

ne perturbe pas le système. La nouvelle action s'intégrera progressivement dans la Q-table, sans que nous perdions le comportement appris jusque là.

Les actions élémentaires dont nous disposons sont les suivantes :

- Transférer un rappel à l'utilisateur. Nous disposons de plusieurs modalités pour cette action, la modalité choisie étant le paramètre de l'action. Le choix de la modalité s'appuie sur le contexte de l'utilisateur (y compris les ressources disponibles) et sur ses préférences. Nous pouvons transférer le rappel par synthèse vocale, en utilisant des haut-parleurs se trouvant dans la même pièce que l'utilisateur. Nous pouvons lui afficher un message écrit sur un écran qu'il serait susceptible de remarquer, y compris son appareil mobile (téléphone ou PDA). Enfin, nous pouvons simplement envoyer un mail à l'utilisateur.
- Informer l'utilisateur d'un nouveau mail qu'il vient de recevoir. Nous disposons également de différentes modalités : la synthèse vocale ou bien un message écrit.
- Verrouiller l'écran de l'utilisateur.
- Déverrouiller l'écran de l'utilisateur.
- Mettre en pause la musique qui joue sur l'ordinateur de l'utilisateur.
- Reprendre la lecture de la musique.
- Ne rien faire est également une action.

L'ensemble des actions de l'assistant est l'ensemble formé par toutes les combinaisons des actions possibles pour chacune des quatre parties suivantes : que faire à propos

1. du rappel ;
2. du nouveau mail ;
3. de l'écran ;
4. de la musique.

Pour chacune de ces parties, il y a trois possibilités : faire quelque chose (par exemple transférer ou verrouiller), faire l'inverse (ne pas informer du mail, relancer la musique) ou bien ne rien faire.

Chaque action est définie avec un attribut indiquant si cette action modifie ou non l'environnement. Par exemple, « ne pas informer l'utilisateur d'un mail » ne modifie pas l'état de l'environnement, ce qui n'est pas le cas de « verrouiller l'écran ».

Ayant défini les actions, nous sommes en mesure de présenter un extrait de la Q-table : la figure 5.5 montre les Q-valeurs de toutes les actions possibles pour l'état suivant :

```
alarm(minute=<*>, title=<*>, hour=<*>) ;
xActivity(isActive=<true>, machine=<+>) ;
inOffice(office=<+>, user=<+>) ;
absent(user=<*>) ;
hasUnreadMail(from=<*>, to=<*>, body=<*>, subject=<*>) ;
entrance(isAlone=<*>, friendlyName=<*>, btAddress=<*>) ;
exit(isAlone=<*> friendlyName=<*>, btAddress=<*>) ;
task(taskName=<*>) ;
user(login=<+>) ;
userOffice(office=<+>, login=<+>) ;
userMachine(login=<+>, machine=<+>) ;
computerState(machine=<+>, isScreenLocked=true, isMusicPaused=true) ;
```

Action	Q-valeur
unpauseMusic&&unlockScreen	17,96
unpauseMusic&&lockScreen	0,48
pauseMusic&&unlockScreen	0,30
notForward&&notInform	0,00
pauseMusic&&nothingAboutScreen	-0,02
unpauseMusic&&nothingAboutScreen	-0,20
nothingAboutMusic&&nothingAboutScreen	-0,48
nothingAboutMusic&&unlockScreen	-2,47
pauseMusic&&lockScreen	-10,82
nothingAboutMusic&&lockScreen	-44,20
notForward&&notInform&&nothingAboutMusic&&nothingAboutScreen	-62,85

FIG. 5.5 – Un extrait de la Q-table pour l'état ci-dessus, dans lequel l'utilisateur est simplement présent dans son bureau. Nous pouvons constater que la meilleure action est de déverrouiller l'écran et de jouer la musique.

Un extrait plus complet est fourni en annexe C.1.

5.6.4 Collecte des récompenses

L'apprentissage par renforcement se base sur les récompenses (ou renforcements) que l'utilisateur donne pour des actions que l'assistant décide d'exécuter. L'apprentissage est d'autant meilleur (rapide et précis) que les renforcements reçus sont fréquents. Mais il est fort probable que l'utilisateur ne donne pas un renforcement pour chaque action, simplement parce qu'il sera trop occupé ou n'y pensera pas. D'après [Richard et Yamada, 2007], les utilisateurs sont souvent réfractaires à fournir des informations précises ou une récompense explicite à un système d'apprentissage. De plus, comme le mettent en avant [Isbell *et al.*, 2001], les récompenses données par l'utilisateur peuvent souvent être incohérentes et se décaler dans le temps :

> « Individual users may be inconsistent in the rewards they provide (even when they implicitly have a fixed set of preferences), and their preferences may change over time (for example, due to becoming bored or irritated with an action). Even when their rewards are consistent, there can be great temporal variation in their reward pattern. »

[Thomaz *et al.*, 2006] ont étudié spécifiquement l'apprentissage par renforcement guidé par des utilisateurs humains non experts. La conclusion de cette étude est que, pour les humains, l'entraînement est un processus à double sens. Les humains ont tendance à vouloir communiquer avec le système, à voir l'entraînement comme un enseignement, un *partenariat*. Les participants ont également montré une tendance à considérer les renforcements plutôt comme une guidance, une indication sur le comportement à avoir dans le futur et non pas une appréciation de la dernière action effectuée.

En outre, les participants ont plus souvent donné des renforcements positifs que négatifs, indépendamment de la qualité de l'apprentissage. Dans le cadre où l'entraînement est perçu comme un enseignement, les humains ont tendance à vouloir motiver l'élève qu'est le système par des récompenses encourageantes. D'autre part, les participants attendaient une amélioration immédiate suite à un

renforcement négatif. Les algorithmes d'AR ne réagissant pas aussi rapidement, les personnes sentaient donc leur retours négatifs ignorés.

Enfin, l'étude a montré une adaptation des personnes au système. Au fur et à mesure de l'interaction, les participants ont construit un modèle mental du système et s'y sont adaptés. Constatant les résultats de leur entraînement, ils ont augmenté la fréquence de leurs renforcements.

Les conclusions de cette étude sont intéressantes pour nous, mais, pour le moment, nous ne savons pas comment en tenir compte. Comment savoir si l'utilisateur donne un renforcement positif parce qu'il est satisfait du système, ou bien parce qu'il essaye de l'encourager ?

Il est possible de se demander si ces participants réagiraient de la même façon en entraînant un système dans le cadre de leur vie quotidienne et à long terme, comme c'est le cas pour notre assistant. Dans l'étude menée par [Thomaz *et al.*, 2006], les participants étaient là clairement pour une expérience et n'avaient qu'une seule tâche en tête : entraîner le robot. Comme le soutiennent [Richard et Yamada, 2007], les utilisateurs d'un système au quotidien seront plus réticents et moins participatifs.

Par conséquent, nous devons adapter notre système au fait que les renforcements donnés seront rares et que, lorsqu'une récompense est donnée, il se peut qu'elle ne concerne pas seulement la dernière action, mais plusieurs actions récentes. Ce dernier point est pris en compte par l'algorithme d'apprentissage (le Q-*Learning* avec traces d'éligibilité, algorithme 2) qui propage en arrière les renforcements.

Pour pallier le manque de renforcements, une idée est de recueillir des renforcements implicites, en plus des renforcements explicitement donnés par l'utilisateur. Ces récompenses implicites proviendraient d'indices tels que la réaction de l'utilisateur quant à la dernière action du système. Prenons par exemple l'action d'informer l'utilisateur d'un e-mail. Si la personne lit l'e-mail immédiatement, alors l'action était probablement appropriée, et le renforcement implicite – positif. Si, au contraire, l'utilisateur ignore le message, alors la récompense déduite est négative. Cependant, ces récompenses implicites devraient avoir une valeur numérique limitée, car elles sont incertaines et ne devraient pas avoir un impact trop important et trop immédiat sur le comportement. Cette solution est préconisée par [Richard et Yamada, 2007] et [Maes, 1994].

Enfin, nous devons tenir compte de l'incohérence éventuelle des renforcements. En effet, l'utilisateur peut être influencé par des éléments que nous ne pouvons pas percevoir lorsqu'il donne une récompense (tels que son humeur ou l'influence d'une tierce personne, ou il peut simplement commettre une erreur). Nous gérons cette possibilité en limitant l'influence d'un renforcement. Ceci sera expliqué en détail section 5.9.2, qui traite de l'apprentissage supervisé du modèle de renforcement. En effet, ce modèle n'est pas totalement modifié par un renforcement contradictoire avec la connaissance actuelle du monde, mais évolue lentement. Si ce renforcement incohérent est une observation aberrante, il ne causera pas de problèmes ; s'il est représentatif d'un changement dans les préférences de l'utilisateur, il se répétera et le modèle s'adaptera peu à peu. Il est d'ailleurs préférable de ne pas modifier radicalement le comportement du système afin de ne pas perturber l'utilisateur (nous mentionnons ceci section 5.2.1).

Les renforcements explicites sont collectés au travers d'une interface graphique montrée figure 5.6. Cette interface est toujours à disposition de l'utilisateur et affiche à tout moment la dernière action exécutée et l'état dans lequel

cette action a été choisie. Le prédicat ayant été mis à jour dans cet état est en gras afin d'être facilement repérable. Dans l'exemple de la figure 5.6, l'utilisateur vient de passer absent et l'action qui a été choisie est de verrouiller l'écran et de mettre la musique en pause. Enfin, l'interface contient un curseur à positionner sur la valeur de renforcement désirée. Puis, le bouton *Set* permet de soumettre cette récompense. L'avantage d'un tel curseur est que les valeurs numériques n'ont pas d'importance. Dans la version finale de cette interface, il serait même inutile de les afficher. L'important est que l'utilisateur donne une récompense relative au minimum, maximum et milieu des valeurs possibles. La valeur exacte n'est pas importante pour l'utilisateur qui positionne le curseur de manière intuitive. [Schiaffino et Amandi, 2004] ont étudié le comportement des utilisateurs face à un système qui leur demande un retour d'information. Les auteurs ont conclu que les usagers sont, en général, d'accord pour donner des retours simples qui ne leur demandent pas beaucoup d'effort, c'est-à-dire une évaluation sur une échelle quantitative ou qualitative. Les utilisateurs sont d'autant plus enclins à fournir ce retour s'ils savent que cela aide à entraîner le système. Par contre leur motivation diminue dans le temps car ils estiment que le système doit avoir fini son apprentissage au bout d'un certain temps.

Remarque : L'interface de la figure 5.6 n'est pas la version définitive, mais plutôt un prototype. En effet, cette interface doit être toujours disponible mais d'une manière non intrusive et non dérangeante. Elle ne devrait pas être visible en permanence, mais plutôt être facilement invoquable par l'utilisateur. De plus, bien que l'état soit compréhensible tel quel, il serait préférable de trouver un moyen de l'afficher d'une manière encore plus lisible. Par exemple, en masquant les prédicats vides ou encore en construisant une phrase à partir des noms des prédicats, des arguments et de leurs valeurs.

5.6.5 Interactions en direct

Dans un système d'apprentissage par renforcement classique, seules les actions de l'agent modifient l'état. Comme nous l'avons mentionné dans les sections 4.5.1 et 5.6.2, l'assistant personnel reçoit des événements des capteurs de l'environnement et en déduit les modifications correspondantes de l'état de l'agent d'AR. Dans notre cas, des *événements extérieurs* dans l'environnement provoquent également des changements d'état. Par exemple, les rappels de l'agenda de l'utilisateur sont détectés par le service `KdeEventsService` et envoyés, formatés en XML, à l'assistant. Celui-ci modifie alors le prédicat `alarm` en remplissant ses arguments avec les valeurs reçues.

L'assistant personnel est placé dans un environnement modifié par des éléments extérieurs sur lesquels il n'a aucun contrôle. L'utilisateur fait partie de ces éléments extérieurs qui agissent sur l'environnement. Ses allées et venues, les e-mails qu'il reçoit, son activité courante, les rappels de son agenda, etc. sont autant d'éléments non déterministes provoquant des changements d'état.

Deux choix se présentent alors. Nous pouvons considérer que l'utilisateur (et avec lui tous les événements imprévisibles) fait partie de l'environnement ou bien qu'il n'en fait pas partie. Il nous est impossible de connaître l'état interne de l'utilisateur. Si celui-ci fait partie de l'environnement, alors l'assistant personnel n'a pas complètement accès à l'état de l'environnement. Nous ne pouvons alors pas modéliser le problème sous la forme d'un PDM, nous nous trouvons dans le cas d'application des PDMPO (introduits section 5.3). Comme

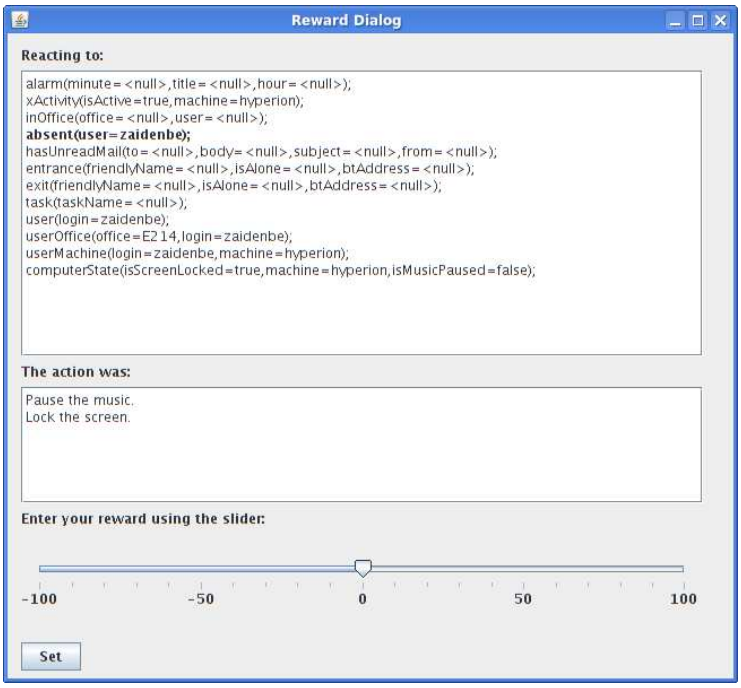

FIG. 5.6 – L'interface de collecte des renforcements.

l'explique [Buffet, 2003], nous avons alors un problème *non-markovien stationnaire*. Le problème sous-jacent est en réalité toujours markovien, mais l'assistant ne peut pas observer le PDM entièrement, il n'a donc accès qu'à un processus non-markovien. Le problème est stationnaire car toute la non-stationnarité introduite par l'utilisateur est déjà prise en compte dans la partie non-observable. L'environnement lui-même peut être considéré comme stationnaire.

Dans le deuxième cas, nous ne considérons pas l'utilisateur comme faisant partie de l'environnement, mais comme un élément extérieur qui perturbe l'environnement de manière non déterministe. L'environnement est désormais constitué de seuls éléments que nous pouvons observer grâce à nos capacités de perception. Étant donné que nous sommes revenus à un environnement entièrement observable, nous sommes revenus à un PDM. L'utilisateur est alors pris en compte par le fait que l'environnement est *non-stationnaire*. Pour reprendre le terme de [Buffet, 2003], nous avons un problème *markovien non-stationnaire*.

Nous avons choisi d'adopter la deuxième manière d'aborder le problème, c'est-à-dire en restant dans le cadre des PDM et en considérant que la loi d'évolution de l'environnement n'est pas stable. Nous pouvons nous permettre ce choix à cause d'une hypothèse très importante : *l'évolution de notre environnement est lente*. En effet, l'environnement est modifié par l'ajout ou le retrait de dispositifs, ou bien la modification du fonctionnement de ces dispositifs. L'hypothèse que ceci n'arrive pas fréquemment est réaliste. Les préférences utilisateur changent également à une faible vitesse. L'utilisateur ne change très probablement pas d'avis tous les jours. L'apprentissage supervisé va intégrer dans le modèle, à partir des exemples, les modifications de l'environnement. Mais il est envisageable de vérifier en plus si le modèle s'applique entièrement aux exemples

récents (observés dans les derniers x jours, semaines voire mois selon la vitesse d'évolution de l'environnement), et de supprimer les parties du modèle qui n'ont plus été observées.

Pour la raison de ce choix, les prédicats que nous avons sélectionnés pour modéliser les états (voir le tableau 5.1) correspondent chacun à un de nos capteurs (décrits section 4.5.1). Ainsi seuls les éléments perçus font partie de l'état, qui est alors entièrement observable. Les autres éléments ne sont pas modélisés et sont traités comme des événements imprévisibles.

Notre problème est similaire au problème d'apprentissage par renforcement dans un cadre multi-agent. Les agents apprennent tous simultanément à agir dans un même environnement, sans se connaître mutuellement. Chaque agent perçoit donc les autres agents comme des perturbations de l'environnement. Dans notre cas, il n'y a qu'un agent mais l'on peut voir l'utilisateur comme l'autre agent perturbant l'environnement. C'est le cadre étudié par [Buffet, 2003].

La non-stationnarité de l'environnement est souvent traitée en laissant une part d'adaptation dans le comportement de l'agent [Sutton, 1990]. Au lieu de progressivement faire décroître le taux d'apprentissage α du Q-Learning (algorithme 1) jusqu'à zéro, on ne le décroît que jusqu'à une limite non nulle, dont la valeur reflète la vitesse d'évolution de l'environnement.

Cette non-stationnarité a plusieurs conséquences. Elle implique que nous ne savons pas tout de l'environnement et de l'utilisateur, et qu'ils peuvent toujours évoluer. Nous avons déjà abordé ce point dès la section 2.3. En effet, nous appliquerons un apprentissage *à vie*, ce qui nous permet de prendre en compte les évolutions de l'environnement et de l'utilisateur. Pour l'environnement, il s'agit de l'apprentissage supervisé des modèles dont il sera question section 5.9. De plus, l'environnement étant non déterministe, son modèle est, par conséquent, *probabiliste*. Ce point sera abordé section 5.7.

D'après [Sutton, 1990], l'avantage de DYNA est justement de pouvoir être appliquée à des environnements stochastiques. La planification peut être effectuée sur des modèles incomplets, changeants et probablement incorrects, construits par apprentissage. Pour pallier ces problèmes, différentes techniques peuvent être appliquées [Sutton et Barto, 1998]. Notamment, il est possible de garder en mémoire le nombre de pas effectués depuis la dernière fois qu'un exemple a été observé dans le monde réel. Dans la phase de planification, pour tester ces actions obsolètes, un renforcement « bonus » est donné aux expériences simulées impliquant ces actions. En effet, un état qui n'a pas été observé depuis longtemps peut avoir évolué dans le monde réel. Par conséquent, il est nécessaire de le re-tester.

Si nous choisissons de considérer notre cas comme un problème *non-markovien* stationnaire, le formalisme du PDM ne s'applique plus et il est nécessaire de modéliser le problème sous la forme d'un PDMPO, voire d'un DEC-PDMPO (un PDMPO décentralisé), dont il est question section 5.3. L'apprentissage dans un PDMPO se fait en utilisant d'autres algorithmes que ceux classiquement utilisés en AR [Singh *et al.*, 1994]. Il est reconnu que cet apprentissage est un problème difficile [Jaulmes *et al.*, 2005, Kaelbling, 2004, Seuken et Zilberstein, 2008].

En particulier, [Papadimitriou et Tsitsiklis, 1987] ont montré que la résolu-

tion exacte d'un PDMPO est un problème PSPACE-*difficile*[15] La résolution d'un PDM à horizon fini est, par contre, un problème P-*complet*. Enfin, le PDM à horizon infini ou non-stationnaire présente un problème de la classe NC.

[Bernstein *et al.*, 2002] ont montré que la résolution d'un DEC-PDMPO, ainsi que d'un DEC-PDM, est un problème NEXP-*difficile*. Dans le cas où l'horizon est inférieur au nombre d'états du modèle, le problème devient NEXP-*complet* et peut donc, en pratique, requérir un temps doublement exponentiel. [Bernstein *et al.*, 2009] soutiennent que la résolution optimale d'un PDMPO ou d'un DEC-PDMPO à horizon infini est un problème qui peut être, en pratique, insoluble car pouvant nécessiter des ressources infinies. Les auteurs proposent alors une méthode pouvant fournir une solution optimale à ϵ près dans un temps et avec des ressources finis.

D'autres méthodes approximatives existent également. [Jaulmes *et al.*, 2005] proposent notamment une méthode qui gère la non-stationnarité en pénalisant les expériences plus anciennes avec un facteur $\nu \in]0; 1[$.

Malgré les recherches actives dans le domaine des DEC-PDMPO, leur résolution reste un problème très complexe. Les algorithmes optimaux jouent un rôle majoritairement théorique et sont très peu utilisés en pratique [Seuken et Zilberstein, 2008]. La résolution nécessite souvent beaucoup de temps et/ou espace mémoire, ou bien est approximative. De plus, la plupart des solutions ont du mal à passer à l'échelle des problèmes réels [Dibangoye *et al.*, 2009a, Seuken et Zilberstein, 2008].

Bien que le cadre des modèles partiellement observables semble plus adapté à notre système, nous choisissons de l'éviter afin d'éviter de nombreuses difficultés et limiter la complexité de notre système. Nous pouvons nous permettre ceci grâce à l'évolution lente de l'environnement. Nous gérons la non stationnarité par un apprentissage à vie, qu'il s'agisse de l'apprentissage par renforcement du comportement, ou de l'apprentissage supervisé des modèles du monde (section 5.9).

Pour en revenir à notre système, nous avons établi que les changements d'états proviennent de deux sources : les événements extérieurs et les actions prises par l'assistant en réponse à ces événements. Chaque action ne change pas nécessairement l'état. En effet, comme il est décrit section 5.6.3, « ne rien faire » ou bien « ne pas verrouiller l'écran » sont des actions. Pour être plus précis, étant donnée la manière dont notre système est conçu, tous les changements d'états proviennent d'événements car ils sont gérés par l'assistant en réponse à un message reçu de la part d'un capteur. Nous distinguons les événements de source inconnue (donc des événements extérieurs) des événements de source connue, c'est-à-dire provoqués par la dernière action qu'a exécutée l'assistant. Pour cela, chaque action est définie avec un booléen indiquant si l'action modifie ou non l'environnement. Par contre, nous ne savons pas *comment* chaque action modifie l'environnement. Ceci sera appris à partir d'exemples (section 5.9.1).

Le fonctionnement final est résumé par la figure 5.7. Notons s l'état courant de l'agent d'AR. Lorsqu'un événement, noté e, survient dans l'environnement, les modules sensoriels de l'assistant le détectent ①. L'assistant détermine, à partir de la perception, le ou les prédicats modifiés dans l'état, il obtient ainsi le nouvel état de l'environnement, noté s'. Il sait alors que s' est l'état suivant de s. Il lui manque de savoir si cette transition a été provoquée par un événement

[15] La définition de cette notion est donnée en annexe C.2.

extérieur, ou bien par suite d'une action de l'assistant. Par exemple, si l'événement reçu est « écran verrouillé », il se peut que ce soit l'utilisateur qui ait lui-même verrouillé son écran, ou bien l'assistant qui ait exécuté l'action *verrouiller l'écran*. L'assistant résout l'ambiguïté en se basant sur la dernière action qu'il a exécutée, notée a. Si l'attribut booléen de a est à vrai, indiquant que a modifie l'environnement, alors la transition qui vient de s'effectuer est $\{s, a, s'\}$ et cet exemple est enregistré tel quel dans la base de données (dans la table `rl_examples_history` de la figure 5.8) ②. Cette dernière action est aussitôt oubliée pour ne pas perturber le pas suivant. Si, par contre, la dernière action ne peut pas modifier l'environnement, alors il s'agit d'un événement extérieur et c'est la transition $\{s, e, s'\}$ qui est enregistrée dans la base de données[16] ②. Il reste une troisième possibilité : l'action a est exécutée et doit modifier l'environnement. Mais avant que l'événement résultant ne soit reçu, un autre événement indépendant survient et perturbe le système. Nous admettons l'hypothèse que les événements ne surviennent pas assez fréquemment pour que cette situation soit courante. Lorsque ceci arrive, nous verrons dans la section 5.9.1 que l'apprentissage supervisé des modèles prend en compte le fait que certains exemples peuvent ne pas être représentatifs d'un véritable changement dans l'environnement.

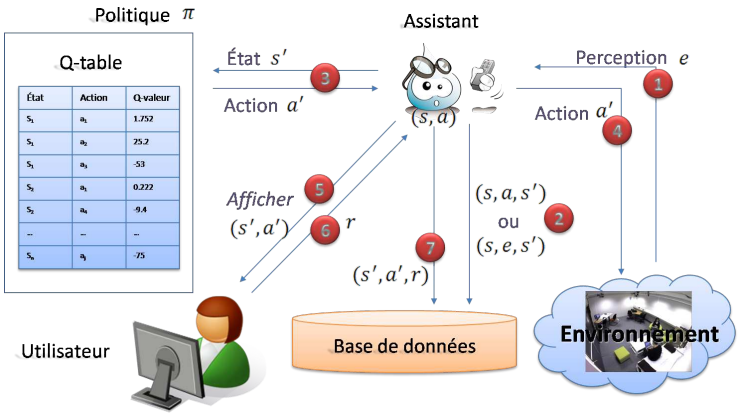

FIG. 5.7 – Les différentes étapes de l'interaction de l'assistant avec l'utilisateur.

S'il s'agit d'un événement extérieur, l'agent d'AR choisit une nouvelle action a' à exécuter en suivant sa politique actuelle ③. Il exécute l'action dans l'environnement en envoyant les commandes nécessaires aux bons modules effecteurs (décrits section 4.5.2) ④. Enfin, l'assistant met à jour les informations de l'interface de récompense (décrite section 5.6.4 et montrée figure 5.6), en affichant l'action a' et l'état s' ⑤. Si l'utilisateur le décide, il invoque cette interface et donne une récompense ⑥. Cette récompense est alors stockée dans la base de données (dans la table `reward_history` de la figure 5.8), sous la forme d'un triplet $\{s', a', r\}$ ⑦.

Les interactions de l'assistant avec l'utilisateur sont donc statiques et n'incluent pas d'apprentissage. L'assistant agit selon son comportement courant et stocke toutes les données sur l'environnement qu'il observe.

[16] La table `rl_examples_history` contient une colonne `is_action` indiquant si le tuple contient une action a ou un événement e.

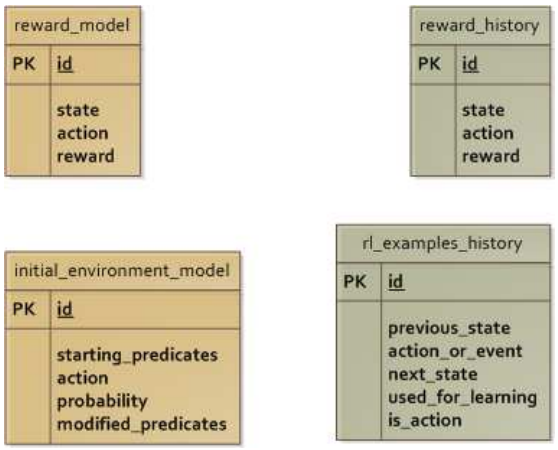

FIG. 5.8 – Les tables de la base de données utilisées pour l'apprentissage par renforcement et l'apprentissage supervisé du modèle du monde.

5.7 Modèle de l'environnement

L'application d'une méthode d'apprentissage par renforcement indirect nécessite un modèle de l'environnement. De telles méthodes proposent d'accélérer l'apprentissage en rejouant virtuellement les expériences vécues. Pour cela il faut disposer d'un monde virtuel dans lequel ces expériences vont être rejouées. La partie gauche de la figure 5.9 représente le modèle classiquement utilisé. Il prend en entrée l'état courant et l'action exécutée dans cet état, et renvoie d'une part le renforcement obtenu et, d'autre part, l'état dans lequel se trouve l'environnement suite à cela. Si ce modèle est probabiliste, alors ce sont les espérances du renforcement et de l'état suivant qui sont retournées. La partie droite de la figure 5.9 représente le modèle que nous utilisons. Il doit être capable de la même chose que le modèle classique, mais également de prendre en entrée l'état courant et un *événement* (au lieu d'une action) et de retourner l'état suivant (aucun renforcement n'est retourné dans ce cas car il n'y a pas lieu de donner un renforcement à un événement extérieur). Ceci est décrit dans la section précédente (section 5.6.5).

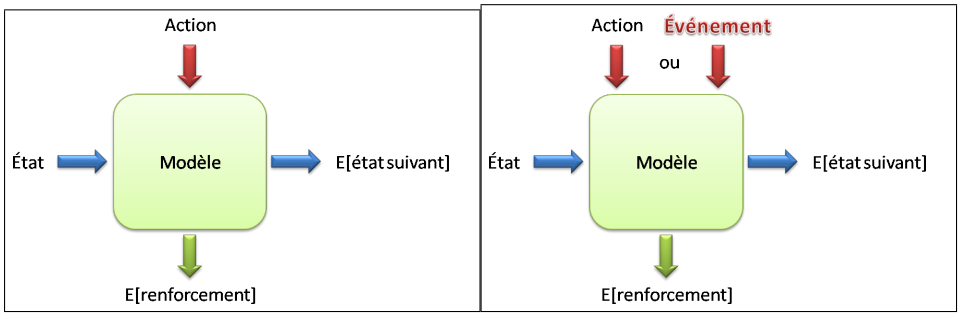

FIG. 5.9 – Les entrées-sorties du modèle du monde. À gauche : le modèle classique ; À droite : notre version du modèle du monde.

D'après [Kaelbling, 2004], un bon modèle du monde devrait être compact, pratique pour la planification et facile à apprendre. D'après l'auteur, la première idée serait d'utiliser les réseaux bayésiens dynamiques. Mais ceux-ci nécessiteraient la spécification de toutes les configurations possibles de toutes les variables du monde, ce qui est trop complexe. Kaelbling suggère donc l'utilisation de règles probabilistes. Ce formalisme fonctionne bien sous les hypothèses que le monde contient des objets dont les propriétés et relations changent dans le temps, que seulement peu d'issues sont possibles après une action et qu'une grande partie de l'état n'est pas modifiée par chaque action. Ces hypothèses s'appliquent à notre cas. Ces règles sont apprises à partir d'expériences dans le monde. Le grand avantage des règles est leur capacité de généralisation. Une seule règle s'applique à tous les objets d'un même type (par exemple).

Le modèle de l'environnement comprend deux éléments : un modèle de transition et un modèle de récompense. Le modèle de transition (ou fonction de transition $\mathcal{P}$) est capable de fournir une estimation de l'état suivant lorsqu'une action a été exécutée dans un certain état de départ. Le modèle de récompense (ou fonction de récompense $\mathcal{R}$) fournit une estimation de la récompense reçue pour avoir exécuté une action donnée dans un état donné.

Étant donné la complexité de l'environnement réel, nous ne pouvons pas prédéfinir ces fonctions. En particulier, la fonction de récompense modélise les récompenses données par l'utilisateur du système donc il nous est impossible de spécifier cette fonction à l'avance[17]. Par conséquent, il est indispensable d'apprendre ces deux modèles à partir des interactions observées. Cet apprentissage doit se baser sur l'environnement réel, il est donc possible d'effectuer un apprentissage supervisé. Les exemples à utiliser en entrée de l'algorithme d'apprentissage supervisé sont enregistrés lors d'interactions réelles du système avec l'environnement et l'utilisateur tel qu'il est décrit section 5.6.5.

5.7.1 Fonction de transition

La fonction de transition $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ est une fonction stochastique markovienne notée $\mathcal{P}(s'|s, a)$. Elle prend en entrée l'état courant et une action, et fournit l'état suivant avec une certaine probabilité. Comme nous l'avons précisé section 5.6.5, nos transitions ne s'effectuent pas seulement suite à une action de l'assistant, mais également suite à un événement extérieur (monde non-stationnaire). Nous redéfinissons donc $\mathcal{P}$ de la façon suivante : $\mathcal{P} : (\mathcal{S} \times \mathcal{A} \times \mathcal{S}) \cup (\mathcal{S} \times \mathcal{E} \times \mathcal{S}) \rightarrow [0, 1]$ où $\mathcal{E}$ est l'ensemble des événements captés par l'assistant. Nous pouvons noter cette fonction $\mathcal{P}(s'|s, o)$ où l'occurrence o est soit une action $a \in \mathcal{A}$, soit un événement $e \in \mathcal{E}$.

Nous avons défini notre fonction de transition comme un ensemble de transformations. Chaque transformation comprend les éléments suivants :

- un état de départ s^t ;
- une occurrence o^t ;
- une probabilité p^t ;
- les modifications à appliquer sur l'état de départ observé afin d'obtenir l'état d'arrivée $M^t = \{m_i^t, i \in [1; n]\}$.

[17] Il est, par contre, envisageable de proposer un questionnaire à l'utilisateur sur les récompenses qu'il donnerait. Ceci n'est possible qu'en se limitant à quelques situations précises et ne nous affranchit pas d'apprendre le modèle complet.

Nous utiliserons la notation suivante pour désigner une transformation : $t(s^t, o^t, p^t, M^t)$.

Lorsque le modèle remplace le monde dans l'apprentissage, tel qu'il est décrit section 5.2.1, le modèle de transition reçoit un état s et une occurrence o et doit retourner l'état suivant s'. Pour cela, il doit trouver une transformation compatible avec les paramètres en entrée (s et o) et calculer l'état à retourner en sortie s' en appliquant les modifications de cette transformation.

D'abord, il recherche dans son ensemble de transformations celles ayant la même occurrence que celle donnée en entrée ($o^t = o$). Si l'occurrence est une action a, alors le modèle recherche les transformations dont l'action a^t est identique à l'action en entrée a ($a^t = a$). Si l'occurrence est un événement e, le modèle cherche plutôt des événements *correspondants* ($e^t \sim e$). En effet, le modèle tend à être le plus générique possible et, tout comme pour les états, la transition sera probablement la même quels que soient les paramètres de l'événement. Par contre, les actions n'ont pas de paramètre, il n'y a donc pas lieu de les généraliser. Ensuite, il compare les états de départ de ces transformations (s^t) avec l'état en entrée s, pour ne garder que celles dont l'état de départ correspond ($s^t \sim s$). Il s'agit bien d'un appariement et non d'une égalité car les états de départ s^t des transformations sont des états génériques.

Une action opère généralement sur une petite partie de l'environnement, correspondant souvent à un ou deux prédicats. Prenons l'exemple de l'action *verrouiller l'écran*. Lorsque cette action est exécutée, quel que soit l'état de départ, dans l'état suivant nous trouverons la valeur `true` pour l'argument `isScreenLocked` du prédicat `computerState`, reflétant le fait que l'écran est verrouillé. La présence ou absence de l'utilisateur, ou encore un éventuel rappel actif n'ont aucune influence sur cette transition. Nous exploitons ce constat pour réduire la taille du modèle (réduire le nombre de transformations). De plus, ceci rend le modèle plus générique car il suffit d'observer cette action une fois dans n'importe quel état pour avoir une transformation opérant dans tous les autres états. Nous verrons en détail dans la suite (section 5.9.1) comment nous gérons les éventuels conflits. Ce principe est identique avec les événements.

Ainsi, l'état de départ s^t est fourni sous forme d'un ensemble de prédicats, sous-ensemble non vide de l'ensemble de tous les prédicats définissant un état. Chaque prédicat fourni contient des valeurs requises pour tous ses arguments. Une valeur requise peut être une valeur spécifique (par exemple pour l'argument `from` du prédicat `hasUnreadMail` on pourrait donner la valeur `directeur@entreprise.com`), y compris la valeur `<null>`, ou bien un caractère générique (`<+>` pour requérir une valeur quelconque mais non vide, ou bien `<*>`, ce qui signifie que toutes les valeurs sont acceptées, y compris `<null>`). Par exemple, nous aurions s^t = `hasUnreadMail(`*`from`*`=directeur@entreprise.com, `*`to`*`=<+>, `*`body`*`=<*>, `*`subject`*`=<*>)`.

Pour savoir si une transformation t peut s'appliquer sur l'état s, nous vérifions que toutes les valeurs requises dans s^t sont satisfaites dans s.

Enfin, il se peut que plusieurs transformations soient applicables. Le modèle tire alors une de ces transformations au hasard en respectant leurs probabilités d'être choisies p^t. La valeur de probabilité de la transformation choisie peut également servir d'indicateur de la confiance du modèle en sa réponse ($\mathcal{P}(s'|s,\ o)$). Si, au contraire, aucune transformation n'est applicable, le modèle retourne l'état s donné en entrée. Dans ce cas, le pas de Q-Learning ne se fait pas

sur cet exemple car ce que renvoie le modèle est clairement faux. Nous évitons d'apprendre quelque chose que nous savons incorrect.

Par exemple, si l'état courant s est l'état suivant :

$s =$
alarm(*minute*=<null>, *title*=<null>, *hour*=<null>) ;
xActivity(*isActive*=true, *machine*=hyperion) ;
inOffice(*office*=E214, *user*=zaidenberg) ;
absent(*user*=<null>) ;
hasUnreadMail(*from*=<null>,*to*=<null>, *body*=<null>, *subject*=<null>) ;
entrance(*isAlone*=true, *friendlyName*=mercure,
btAddress=00:17:03:A4:CA:F2) ;
exit(*isAlone*=<null>, *friendlyName*=<null>, *btAddress*=<null>) ;
task(*taskName*=developpement) ;
user(*login*=zaidenberg) ;
userOffice(*office*=E214, *login*=zaidenberg) ;
userMachine(*login*=zaidenberg, *machine*=hyperion) ;
computerState(*machine*=hyperion, *isScreenLocked*=false,
isMusicPaused=false) ;

Dans cet état, l'utilisateur est présent dans son bureau, la musique est entrain de jouer, et une autre personne entre. L'action choisie est de mettre la musique en pause : $a =$ `nothingAboutScreen&&pauseMusic`. Supposons que deux transformations du modèle, t_1 et t_2, soient alors applicables. Elles sont définies par $a^{t_1} = a^{t_2} = a$ et ont les états de départ suivants :

$s^{t_1} =$
entrance(*isAlone*=true, *friendlyName*=<+>, *btAddress*=<+>) ;
computerState(*machine*=<+>, *isScreenLocked*=false,
isMusicPaused=false) ;

Et $s^{t_2} =$ inOffice(*office*=<+>, *user*=<+>). t_1 s'applique à tous les états dans lesquels quelqu'un entre dans le bureau et l'écran de l'ordinateur de l'utilisateur est déverrouillé, et sa musique n'est pas en pause. t_2 est plus générale puisqu'elle s'applique dès lors que l'utilisateur est présent dans son bureau.

Ce mécanisme d'appariement peut être formalisé par l'algorithme 4. Dans cet algorithme, nous notons un état s comme l'ensemble de ses prédicats : $s = \{p_s\}$ et chaque prédicat, comme l'ensemble de ses arguments : $p_s = \{a_{p_s}\}$. Enfin, la

valeur d'un argument est notée $v(a_{p_s})$.

Algorithme 4 : L'algorithme d'appariement entre l'état courant s et les transformations du modèle de transition.

Entrée : l'état courant s, l'action exécutée a, l'ensemble des transformations du modèle noté $\mathcal{P} = \{t\}$
Sortie : l'ensemble des transformations applicables à s, noté $\mathcal{P}_{\text{app}}$
Initialisation : $\mathcal{P}_{\text{app}} \leftarrow \mathcal{P}$;
Pour chaque $t \in \mathcal{P}$ **faire**
  si $o^t \sim o$ **alors**
    Pour chaque $p_{s^t} \in s^t$ **faire**
      Pour chaque $a_{p_{s^t}} \in p_{s^t}$ **faire**
        si $v(a_{p_{s^t}}) =$ `<*>` **alors**
          si $v(a_{p_s}) =$ `<null>` **alors**
            $\mathcal{P}_{\text{app}} = \mathcal{P}_{\text{app}} \setminus t$
        sinon si $v(a_{p_{s^t}}) \neq$ `<*>` **alors**
          si $v(a_{p_s}) \neq v(a_{p_{s^t}})$ **alors**
            $\mathcal{P}_{\text{app}} = \mathcal{P}_{\text{app}} \setminus t$
  sinon
    $\mathcal{P}_{\text{app}} = \mathcal{P}_{\text{app}} \setminus t$

Il est possible d'associer un coefficient avec chaque transformation représentant son « degré de spécialisation ». Pour chaque valeur appariée, ce coefficient augmente. Si une valeur exacte est appariée, un bonus est ajouté au coefficient. Il est ainsi possible de choisir la transformation la plus spécifique.

L'algorithme 4 fournit l'ensemble des transformations applicables à l'état courant s, dans l'exemple cet ensemble serait $\mathcal{P}_{\text{app}} = \{t_1, t_2\}$. On en choisit une au hasard en fonction de leurs probabilités (p^{t_1} et p^{t_2}).

Une fois la transformation choisie, supposons que ce soit t_1, il suffit de l'appliquer à l'état courant s pour obtenir l'état suivant s'. s' est une copie de s sur laquelle sont appliquées les modifications M^{t_1}. Une modification concerne un argument d'un prédicat et peut être de quatre sortes :

- effacer la valeur (`setNull`) ;
- mettre une valeur donnée (`setValue`) ;
- mettre la valeur d'un argument donné d'un autre prédicat (`setValueOf`) ;
- effacer l'espampille (`resetTimestamp`).

La dernière modification, effacer l'espampille, concerne en réalité un prédicat en entier et non un argument. La première modification, effacer la valeur, consiste à mettre la valeur à `<null>`.

Dans notre exemple, $M^{t_1} = \{m_1^{t_1}\}$ avec $m_1^{t_1} =$ `setValue(computerState:isMusicPaused, true)`), c'est-à-dire que la transformation met à *vrai* la valeur de l'argument `isMusicPaused` du prédicat `computerState`.

Les transformations sont écrites en XML, en suivant le schéma XSD fourni figure 5.10. L'élément racine est une liste de transformations. Nous distinguons clairement les transformations opérant sur les actions de celles opérant sur les

événements, mais leur représentations est tout à fait similaire. Pour plus de clarté, la figure 5.10 ne détaille que les transformations sur les actions.

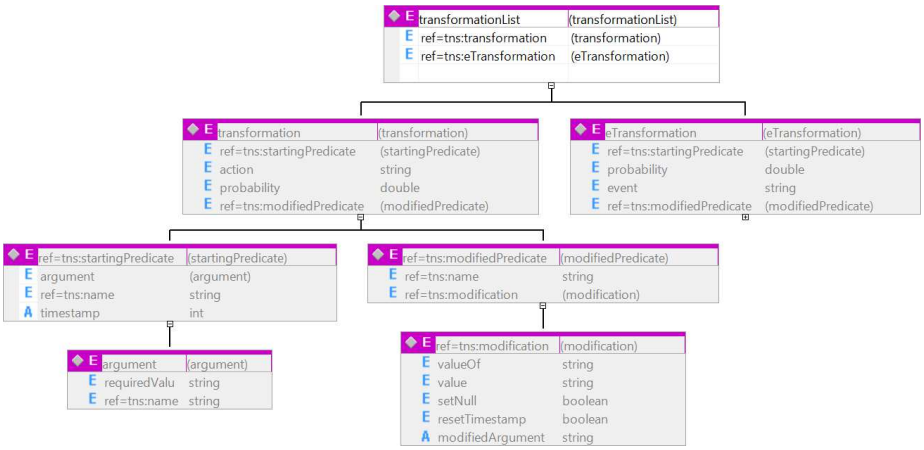

FIG. 5.10 – Le schéma XSD utilisé pour représenter le modèle de transition.

Ainsi, la transformation t_1 de l'exemple précédent s'écrit en XML comme le montre la figure 5.11.

5.7.2 Fonction de récompense

La fonction de récompense immédiate $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ est notée $\mathcal{R}(s,\ a)$ et fournit une valeur de récompense étant donné l'état courant s et l'action a exécutée dans cet état. Il n'y a, de toute évidence, pas lieu de donner une récompense lorsqu'un événement extérieur survient, nous ne modifions donc pas la définition classique de la fonction de récompense, comme nous avons dû le faire avec la fonction de transition.

Tout comme la fonction de transition, la fonction de récompense est un ensemble d'entrées e, chaque entrée étant composée des trois éléments suivants :

- un état de départ s^e ;
- une action a^e ;
- une valeur de récompense r^e.

De même que précédemment, la récompense dépend d'une part de l'action, et d'autre part d'une sous-partie de l'état. La récompense dépend rarement de tous les paramètres définissant l'état. Par exemple, l'utilisateur donnera une très mauvaise récompense si l'assistant verrouille son écran lorsqu'il était entrain de taper sur le clavier, mais la présence d'autres personnes ou d'un nouvel e-mail n'entrent pas en compte dans cette décision. Seul le prédicat `xActivity` importe. Par conséquent, l'état de départ s^e est donné également sous forme d'un sous-ensemble non vide des prédicats existants, et de valeurs requises pour leurs arguments (ces états de départ sont génériques).

Si nous reprenons l'exemple de la section précédente (section 5.7.1), nous aurions une entrée dans le modèle de renforcement qui serait écrite en XML de façon similaire à la transformation de la figure 5.11, mais avec l'élément `<reward>95</reward>` au lieu de l'élément `<modifiedPredicate>`. Lorsque plusieurs entrées du modèle correspondent à l'état courant s, nous calculons le

```
<transformation>
   <startingPredicate>
      <name>entrance</name>
      <argument>
         <name>isAlone</name>
         <requiredValue>true</requiredValue>
      </argument>
      <argument>
         <name>friendlyName</name>
         <requiredValue><+></requiredValue>
      </argument>
      <argument>
         <name>btAddress</name>
         <requiredValue><+></requiredValue>
      </argument>
   </startingPredicate>
   <startingPredicate>
      <name>computerState</name>
      <argument>
         <name>machine</name>
         <requiredValue><+></requiredValue>
      </argument>
      <argument>
         <name>isScreenLocked</name>
         <requiredValue>false</requiredValue>
      </argument>
      <argument>
         <name>isMusicPaused</name>
         <requiredValue>false</requiredValue>
      </argument>
   </startingPredicate>
      <action>nothingAboutScreen&&pauseMusic</action>
      <probability>0.9</probability>
   <modifiedPredicate>
      <name>computerState</name>
      <modification modifiedArgument="isMusicPaused">
         <value>true</value>
      </modification>
   </modifiedPredicate>
</transformation>
```

FIG. 5.11 – Exemple de transformation au format XML.

renforcement final à partir des renforcements de ces entrées. Nous faisons une simple moyenne, mais une meilleure technique serait de privilégier les entrées qui correspondent de la manière la plus spécifique à l'état courant. Enfin, le modèle retourne zéro si aucune entrée correspondante n'est trouvée.

5.8 Injection de connaissances initiales

Parmi les contraintes définies pour atteindre notre objectif (section 2.4) figure celle de créer un système fonctionnel dès le départ. Avant que l'assistant ne commence à agir, il doit avoir un comportement qui n'est pas incohérent. Néanmoins, spécifier des Q-valeurs initiales est difficile. D'après [Kaelbling, 2004], il est plus facile de spécifier un renforcement qu'un comportement. Il est même envisageable de proposer à l'utilisateur de spécifier des préférences initiales en termes de renforcements. De plus, décrire en partie l'environnement n'est pas une tâche très complexe.

Par ailleurs, la Q-table n'est pas le formalisme central de notre système. Elle peut être déduite du modèle de contexte. Par conséquent, ce modèle est, lui, l'élément central de l'assistant. Il peut également servir de pivot pour exprimer les mêmes connaissances dans d'autres formalismes. Contrairement à la Q-table, il n'est pas spécifique à une représentation. Si nous décidons d'étendre notre système en ajoutant d'autres mécanismes d'apprentissage (par exemple l'apprentissage par analogie ou bien un apprentissage par renforcement basé sur d'autres types de PDMs, tels que les RMDPs, etc.), le modèle du monde pourra être réutilisé tel quel par ces autres algorithmes, alors que les Q-valeurs n'ont pas de sens dans un autre contexte.

Pour initialiser le système, nous préférons donc fournir un modèle du monde initial plutôt qu'une Q-table initiale. Ce modèle par défaut est très partiel, il ne contient que les informations les plus immédiates (par exemple, lorsque l'action est de verrouiller l'écran, l'écran est verrouillé dans l'état suivant). [Kaelbling, 2004] pense également que lorsqu'il s'agit d'un système réel, apprendre à partir de zéro est très difficile. Une aide minime de la part de l'homme est un apport énorme pour l'apprentissage.

Afin de « convertir » ce modèle en un comportement, il est nécessaire d'effectuer un premier apprentissage. La section 6.4 traite de cet apprentissage initial. Nous utilisons la base de données pour stocker ce modèle initial, plus exactement les tables `initial_environment_model` de la figure 5.8 pour le modèle de transitions et `reward_model` pour le modèle de récompense. Cette table est ensuite complétée avec le modèle au fur et à mesure que celui-ci est appris (section 5.9.2).

Remarque : dans la conception de notre système, rien n'empêche l'expert ou l'utilisateur de fournir une connaissance initiale aussi étendue qu'il le souhaite. L'apprentissage nous donne un moyen de nous en affranchir, mais ne supprime pas cette possibilité. Si les connaissances initiales sont très riches, nous obtenons un système similaire au produit *Mir:ror* décrit section 4.1.1.1. Ce système commercial permet à l'utilisateur de définir son modèle de contexte en associant des actions à des tags RFiD collés sur les objets de son choix. Approcher un objet d'un lecteur RFiD active alors un service.

5.9 Apprentissage supervisé du modèle de l'environnement

Comme il a été mentionné (notamment dans les sections 5.2.1 et 5.6.5), l'environnement est trop complexe pour être modélisé manuellement. De plus, il peut évoluer et notre système serait rapidement dysfonctionnel. Pour éviter

à un expert ou à l'utilisateur de devoir actualiser le modèle, nous préférons l'apprendre de manière supervisée à partir d'exemples enregistrés pendant les interactions réelles.

L'algorithme d'apprentissage supervisé a pour but de mettre à jour le modèle si les exemples montrent que le monde a évolué. La mise à jour ne doit cependant pas être radicale car un exemple pourrait être aberrant. Nous avons mentionné cette possibilité pour le modèle de transitions dans la section 5.6.5. En ce qui concerne le modèle de récompenses, nous avons exposé dans la section 5.6.4 que les récompenses de l'utilisateur peuvent être incohérentes. Entre autres, l'utilisateur peut être influencé par des éléments extérieurs au moment de donner sa récompense. Il pourrait être dans un état émotionnel particulier, etc. Nous n'avons qu'une perception partielle de l'utilisateur. Pourtant, celui-ci donne une récompense en fonction de sa propre perception de l'environnement et de lui-même, qui est bien plus riche que la nôtre. Si une récompense donnée ne correspond pas au modèle, il ne faut pas modifier le modèle radicalement afin de coller à cette récompense car elle pourrait être incohérente. Par contre, il faut modifier le modèle graduellement car cette récompense pourrait refléter un changement dans les préférences et habitudes de l'utilisateur. Il en est de même pour le modèle de transition. Un exemple de transition peut toujours être erroné, il peut venir de bruit dans les capteurs, d'une erreur de détection ou encore d'une perturbation ponctuelle de l'environnement. Nous ne devons donc pas radicalement modifier le modèle, mais le modifier progressivement dans cette direction et attendre d'autres exemples qui confirmeront ce changement. Nous supposons que l'environnement, ainsi que les préférences utilisateur, ne subissent pas de changement radicaux et fréquents, donc le fait de ne pas être totalement réactif n'est pas pénalisant. Par contre, être instable serait beaucoup plus pénalisant par rapport à l'utilisateur.

5.9.1 Apprentissage supervisé de la fonction de transition

Cet apprentissage s'effectue sur les exemples de transitions enregistrés durant les interactions réelles et qui sont de la forme $\{s, o, s'\}$, où o est une occurrence pouvant être une action a ou bien un événement e. Les états s et s' sont des états ayant été perçus durant les interactions (il en est de même des événements

e, lorsque o est un événement). Il s'agit de l'algorithme 5.

Algorithme 5 : L'algorithme d'apprentissage supervisé du modèle de transition.

Entrée : Un ensemble d'exemples $\{s, o, s'\}$
Sortie : $\mathcal{P}$
Pour chaque *exemple* $\{s, o, s'\}$ **faire**
  si *une transformation t permettant d'obtenir s' à partir de s avec l'occurrence o, peut être trouvée* **alors**
    Augmenter la probabilité de t;
  sinon
    Créer une transformation partant de s, ayant l'action a, ou bien un événement générique crée à partir de e, et se terminant dans s', avec une faible probabilité;
    Diminuer la probabilité de toute autre transformation t' qui correspond à l'état de départ s et a l'occurrence o mais dont l'état d'arrivée est différent de s';

Cet algorithme permet d'une part de renforcer le modèle lorsque de nouveaux exemples confirment les connaissances déjà incluses dans le modèle (par l'augmentation de la probabilité de la transformation qui vient d'être confirmée par un nouvel exemple). D'autre part, il permet de mettre à jour le modèle en intégrant les exemples jamais encore observés. Si un nouvel exemple contredit le modèle existant, nous diminuons le poids des connaissances du modèle. Si cet exemple correspond à un changement dans l'environnement, alors il se reproduira et fera basculer le modèle (la probabilité de l'ancienne transformation deviendra plus petite que celle de la nouvelle, jusqu'à devenir négligeable). Si cet exemple est une aberration, alors le modèle ne sera pas déstabilisé et la probabilité diminuée sera ré-augmentée à terme.

La capacité de généralisation du modèle réside dans la création des nouvelles transformations. Lorsqu'un exemple $\{s, o, s'\}$ n'est « expliqué » par aucune transformation dans le modèle, l'algorithme crée une nouvelle transformation t à partir de cet exemple. L'état de départ de t, s^t, est l'état générique obtenu à partir de s, de la même façon que nous obtenons les états génériques de la Q-table (section 5.6.1.2). Nous pouvons nous permettre d'effectuer cette généralisation car l'issue d'une action dépend uniquement de l'état de l'environnement dont nous avons une représentation suffisamment précise. Si o est une action a, celle-ci est copiée dans la transformation ($a^t = a$). Si o est un événement, alors celui-ci est généralisé (les paramètres non nuls sont remplacés par le joker `<+>`, les valeurs booléennes sont laissées telles quelles, et les autres paramètres sont remplacés par le joker `<*>`) : $o^t = \text{generalisation}(e)$. Enfin, les modifications permettent de transformer l'état de départ s de l'exemple vers l'état suivant s' de l'exemple. Les particularités de l'état de départ que la généralisation efface se retrouvent dynamiquement dans les modifications à appliquer pour obtenir l'état suivant qui gardera donc ces spécificités. Comme la section 5.7.1 l'explique, les modifications indiquent comment construire l'état suivant à partir de l'état courant. Elles peuvent être de la forme `valueOf(pred:arg)`(pour un argument précis), ce qui signifie que cet argument prend la valeur de l'argument `arg` du

prédicat `pred` dans l'état de départ.

Afin de vérifier que le modèle appris a effectivement cette capacité de généralisation, nous avons effectué une validation croisée de cet algorithme. Celle-ci est présentée section 6.6.

5.9.1.1 Division de transformations

Tout comme pour les états dans la Q-table, il est excessif de généraliser toutes les transformations. Il serait judicieux d'appliquer un mécanisme de division de transformations similaire au mécanisme de division d'états évoqué section 5.6.1.3. En effet, il pourrait y avoir des cas dans lesquels l'état suivant dépend des valeurs exactes d'un événement, bien que nous n'ayons pas d'exemple de ceci dans notre application. Une telle situation pourrait se produire par exemple si nous distinguions les entités du système. On pourrait imaginer un robot mobile tel qu'*Aibo*, le chien robot de Sony montré figure 4.2, faisant partie du système. Lorsqu'*Aibo* entre dans une pièce, un événement Bluetooth comme les autres est envoyé. Pourtant, il serait envisageable de distinguer l'entrée d'un humain et d'un robot en ajoutant un prédicat spécial pour la présence d'un robot. L'événement d'entrée d'un périphérique Bluetooth aurait donc deux issues différentes dans l'état suivant, selon l'adresse Bluetooth contenue dans l'événement.

Un post-traîtement pourrait révéler ces cas spéciaux. Une piste (similaire à ce qui est décrit section 5.6.1.3) serait d'effectuer une spécialisation en analysant l'historique des transitions et en appliquant le modèle sur chacune d'elles. Si, pour certains exemples, le modèle se trompe systématiquement, l'on pourrait alors construire des transformations contenant les valeurs exactes de l'événement au lieu de valeurs génériques.

5.9.2 Apprentissage supervisé de la fonction de récompense

Le modèle de récompense est appris sur les récompenses que l'utilisateur a données lors d'interactions réelles du système avec l'environnement. Durant ces interactions, l'assistant enregistre tous les exemples de la forme $\{s, a, r\}$ et l'algorithme 6 permet d'apprendre de manière supervisée un modèle à partir de

ces exemples.

Algorithme 6 : L'algorithme d'apprentissage supervisé du modèle de renforcement.

Entrée : Un ensemble d'exemples $\{s, a, r\}$
Sortie : $\mathcal{R}$
Pour chaque *exemple* $\{s, a, r\}$ **faire**
  si *une entrée du modèle* $e = \{s_e, a_e, r_e\}$ *telle que* $s \sim s_e$ *et* $a = a_e$ *peut être trouvée* **alors**
    Mettre à jour e, assigner $r_e = mix(r, r_e)$, où mix est une fonction de combinaison;
  sinon
    Ajouter une nouvelle entrée $e = \{s, a, r\}$ au modèle de récompense;

Contrairement au modèle de transition (section 5.9.1), nous ne généralisons pas les états lors de l'apprentissage. Des états génériques peuvent être présents dans le modèle, mais ils font partie du modèle initial fourni par un humain, ou bien, il serait envisageable de laisser la possibilité à l'utilisateur d'ajouter explicitement des entrées a posteriori.

Le modèle de récompense se prête mal à la généralisation automatique. En effet, nous n'avons pas accès à l'état interne de l'utilisateur, par conséquent nous ne comprenons pas ses motivations pour donner tel ou tel renforcement. Si nous généralisons l'état, nous risquons donc de perdre de l'information. Dans le cas du modèle de transition ceci est moins vrai, comme nous l'avons évoqué ci-dessus (section 5.9.1). L'issue d'une action ne dépend pas de l'état interne de l'utilisateur, mais uniquement de l'état de l'environnement dont nous avons une représentation plus précise. De plus, les particularités de l'état de départ que la généralisation efface se retrouvent dynamiquement dans les modifications à appliquer pour obtenir l'état suivant qui gardera donc ces spécificités.

La fonction de combination mix utilisée dans l'algorithme 6 doit fusionner un renforcement du modèle avec un renforcement reçu. Actuellement, nous utilisons $mix(r, r_e) = 0.3\,r + 0.7\,r_e$. Ceci traduit le poids du nouvel exemple par rapport au modèle. Attribuer un poids faible au nouvel exemple permet de ne pas radicalement changer le modèle à cause d'un seul exemple car celui-ci pourrait être aberrant. Tout comme pour le modèle de transition (section 5.9.1), le modèle doit évoluer graduellement afin d'être stable. Si r est très différent de r_e parce que l'utilisateur a modifié ses préférences, alors nous aurons d'autres exemples de cette tendance et à terme, le modèle correspondra à l'utilisateur.

5.10 Apprentissage par renforcement non interactif

L'apprentissage à proprement parler est entièrement non interactif. Il consiste à utiliser le modèle du monde dans son état actuel pour mettre à jour la politique, en passant bien sûr par la mise à jour de la Q-table. Il s'agit de jouer des *épisodes* d'AR. La figure 5.12 résume cet algorithme.

Chaque épisode débute dans un état initial. Cet état initial peut être choisi de trois manières : on peut utiliser un état vide (tous les arguments étant à `<null>`) ou bien un état généré aléatoirement, ou encore tiré au hasard parmi les états enregistrés dans la base de données, c'est-à-dire un état déjà observé. La première option n'a pas vraiment d'intérêt. La deuxième permet d'augmenter l'exploration de l'espace d'états et d'essayer de passer par des états inconnus afin d'acquérir une estimation du comportement à avoir lorsqu'ils seront rencontrés réellement. Ceci a un sens car notre modèle de transition a une capacité de généralisation (voir section 5.9.1). En effet, le modèle appris est plus général que les exemples d'interactions réelles enregistrés et sur lesquels il a été appris. La troisième option permet de renforcer l'expérience vécue. Un épisode se termine au bout de k itérations, ou bien lorsqu'un état final a été atteint. Notre application n'a pas d'états finaux car l'assistant n'a pas de but spécifique à remplir, seulement la maximisation des récompenses. Il est tout de même possible de trouver un autre critère d'arrêt prématuré d'un épisode basé sur la mise à jour de la Q-table. Lorsque la différence des Q-valeurs entre deux pas n'est plus suffisamment significative, nous pouvons arrêter l'épisode en cours.

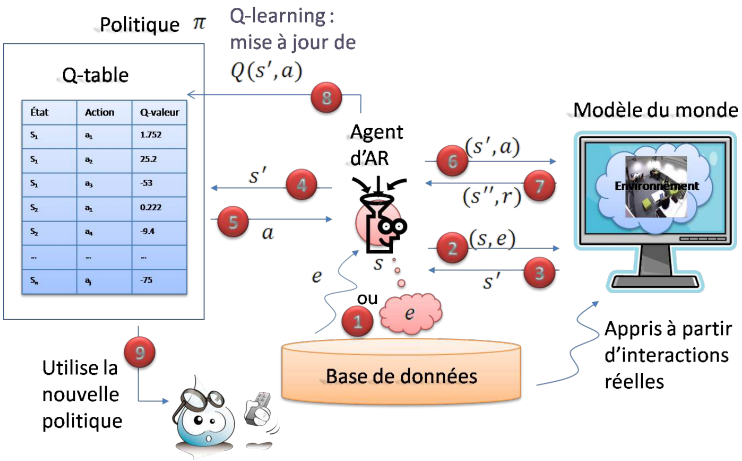

FIG. 5.12 – Les étapes d'un pas de Q-Learning non interactif, utilisant le modèle du monde.

Au cours de l'épisode, ce sont les événements qui génèrent les changements d'état car ceci correspond à ce qu'il se passe dans la réalité[18]. Il existe également deux stratégies : générer des événements aléatoires ou bien tirer au hasard un événement déjà observé. Les deux options se différencient de la même façon que pour les états initiaux : générer des événements aléatoires permet d'explorer d'avantage l'espace d'états car il se pourrait qu'un événement aléatoire nous amène dans un état non encore visité, alors que rejouer des événements de la base de données permet de renforcer l'expérience vécue, d'apprendre autant en ayant observé un événement qu'une seule fois, que ce que l'on aurait appris en

[18]Pour être plus précis, dans la réalité les actions génèrent également des changements d'état, mais par l'intermédiaire des événements qu'elles provoquent. Ces événements, eux, sont la source des changements d'état. Dans les épisodes virtuels il suffit de considérer les événements pour générer des changements d'état car il n'y a plus de différence entre un événement qui s'est produit « tout seul », et un événement provoqué par une action.

observant l'événement n fois. L'utilisateur n'a alors pas besoin de recevoir n e-mails pour que le système ait un comportement cohérent, un seul e-mail suffit à avoir une bonne estimation du comportement. Ceci accélère donc l'apprentissage mais suppose d'avoir un modèle du monde de qualité suffisante.

Notons s l'état courant au début d'une itération. La première étape consiste à choisir un événement e (étape ① de la figure 5.12). Puis, l'agent d'apprentissage par renforcement interroge le modèle du monde ② (le modèle de transitions plus particulièrement) pour connaître l'état suivant s' provoqué par cet événement ③. Si le modèle n'est pas capable de répondre à cette question, l'itération courante est abandonnée. Si le nouvel état est bien renvoyé, l'agent d'AR utilise sa politique ④ qui choisit l'action à exécuter a ⑤. Ayant un couple état action (s', a), l'agent interroge le modèle du monde ⑥ qui lui répond par l'état suivant et le renforcement reçu (s'', r) ⑦. À cette étape, l'agent d'AR a tous les éléments pour exécuter un pas de Q-Learning $\langle s', s'', a, r\rangle$ ⑧ et mettre à jour la Q-table. Ceci clos l'itération courante[19]. La dernière étape ⑨ consiste à remplacer la Q-table utilisée par l'assistant lors d'interactions réelles par la nouvelle Q-table mise à jour. Comme il a été mentionné dans la section 5.2.1, nous avons peut-être intérêt à ne pas utiliser le nouveau comportement dès la fin de chaque épisode afin de ne pas perturber l'utilisateur. Nous pensons que le mieux est de laisser le choix à l'utilisateur de spécifier au bout de combien de temps ou d'épisodes il faut utiliser la nouvelle Q-table.

Ceci est traduit par l'algorithme 7.

Algorithme 7 : Un épisode de Q-Learning utilisé pour l'étape de planification par l'agent d'AR.

Entrée : $\mathcal{P}$, $\mathcal{R}$
pour i *de* 1 *à* k **faire**
  Choisir un événement e;
  Envoyer l'état courant s et l'événement e au modèle de l'environnement et obtenir une prédiction sur l'état suivant s' :
  $s' \leftarrow max_{s' \in \mathcal{S}} \mathcal{P}(s'|s, e)$;
  Choisir une action $a = \pi(s')$;
  Envoyer s' et a au modèle de l'environnement et obtenir des prédictions sur l'état suivant s'' et la récompense r :
  $s'' \leftarrow max_{s'' \in \mathcal{S}} \mathcal{P}(s''|s', a)$, $r \leftarrow \mathcal{R}(s', a)$;
  Appliquer une méthode d'apprentissage par renforcement à l'expérience hypothétique $\langle s', s'', a, r\rangle$:
  $Q(s', a) \leftarrow Q(s', a) + \alpha(r + \gamma \max_{a'} Q(s'', a') - Q(s', a))$;
  $i = i + 1$;

[19] Dans la réalité, la suite serait l'exécution de l'action et la réaction à l'éventuel événement que cette action provoquerait. Nous ne restituons pas cette étape car le quadruplet $\langle s', s'', a, r\rangle$ est suffisant pour effectuer le pas d'apprentissage. Nous supposons que l'événement qu'aurait éventuellement généré l'action sera, à terme, choisi au hasard et ainsi pris en compte.

5.11 Exemple d'illustration – troisième niveau de précision

À ce point, nous sommes en mesure de reprendre le scénario exemple illustrant le fonctionnement de notre système en expliquant tous les détails. Dans la section 2.5, nous avons décrit ce scénario en surface. Dans la section 4.6, nous avons pu y ajouter des aspects techniques sur la communication entre différents modules de l'environnement car nous avions alors présenté notre système ambiant. Ici, nous pouvons entrer encore plus dans les détails et exposer les aspects liés à la représentation de l'agent d'apprentissage par renforcement.

L'agenda de l'utilisateur déclenche un rappel. L'assistant reçoit cet événement. Il sait percevoir le nouvel état résultant cet événement, il modifie donc le prédicat correspondant (`alarm`) et remplit ses arguments avec les paramètres de l'événement qu'il vient de recevoir (heure et titre du rappel) : par exemple `alarm(title=Réunion de thèse, hour=15, minute=30)`. L'assistant transmet ce changement d'état (le prédicat modifié, dont l'estampille a par ailleurs été remise à zéro) à l'agent d'apprentissage par renforcement. Celui-ci s'occupe de sauvegarder ce qui vient de se passer dans la base de données (il s'agit d'un exemple $\{s, e, s'\}$). Puis, il interroge sa politique afin de choisir l'action à exécuter ($a = \pi(s, s')$).

Supposons que l'action choisie soit de transmettre le rappel à l'utilisateur. Dans ce choix intervient également le reste de l'état. Ainsi, l'agent AR transmet à l'assistant l'action à exécuter, qui s'occupe de l'exécuter dans l'environnement, comme le décrit la section 4.6, en prenant en compte les connaissances disponibles sur la localisation actuelle de l'utilisateur et sa modalité préférée (qui dépend d'éléments du contexte tels que la présence d'autres personnes par exemple). Si un renforcement de l'utilisateur a pu être recueilli (de manière directe à travers l'interface graphique, ou bien de manière indirecte), alors l'agent AR l'enregistre dans la base de données : $\{s, a, r\}$. En parallèle de ceci, des épisodes d'apprentissage (algorithme 7) sont exécutés à intervalles de temps réguliers.

5.12 Conclusion

Dans ce chapitre, nous avons proposé une solution au problème de l'apprentissage d'un modèle de contexte dans le cadre de l'informatique ubiquitaire. Un environnement équipé d'un système ambiant détecte le contexte de l'utilisateur et peut agir en fonction de cette observation. Mais comment trouver l'action optimale pour chaque situation et chaque utilisateur ? Nous avons proposé d'apprendre automatiquement ce choix en se basant sur un entraînement de la part de l'utilisateur. L'apprentissage par renforcement convient car l'entraînement est simple – une récompense positive ou négative.

Nous avons alors appliqué l'apprentissage par renforcement à l'environnement ubiquitaire, non sans l'adapter à ce cadre spécifique. Nous avons proposé une modélisation des états de l'environnement qui est compréhensible pour les utilisateurs. Le système peut ainsi montrer ses représentations internes aux usagers. Un système transparent est plus enclin à gagner la confiance de l'utilisateur. De plus, nous avons effectué une généralisation des états sans laquelle l'espace d'état aurait une taille trop grande pour être exploitable.

Pour aborder le problème de la lenteur de l'apprentissage par renforcement à partir des seules interactions avec l'utilisateur, nous avons appliqué l'apprentissage par renforcement indirect. Celui-ci requiert un modèle de l'environnement. Nous avons proposé un tel modèle, et deux algorithmes d'apprentissage supervisé pour l'apprendre à partir d'interactions réelles. Une partie de ce modèle concerne les transitions entre les états de l'environnement. Ce modèle de transition est également générique à un certain degré afin de décrire plus efficacement le monde réel.

Finalement, l'apprentissage est entièrement non interactif, de manière indirecte. En ligne, les interactions sont enregistrées pour l'apprentissage des modèles, servant eux-mêmes à l'apprentissage par renforcement indirect. Ceci nous permet de maîtriser le moment où le nouveau comportement appris remplace l'ancien, en fonction des préférences de l'utilisateur.

Chapitre 6

Mise en place des expériences

Dans les chapitres précédents, nous avons présenté le système que nous avons mis en place afin de rendre notre environnement ambiant (chapitre 4) et l'assistant personnel que nous avons créé afin d'apprendre les préférences de l'utilisateur par rapport au comportement du système (chapitre 5). Nous allons maintenant aborder l'évaluation de cet assistant. Au travers de plusieurs expériences de complexité croissante, nous allons mesurer les performances de nos algorithmes d'apprentissage. Dans la suite de ce chapitre, nous allons décrire ces expériences, puis, dans le chapitre suivant (chapitre 7), nous présenterons les résultats de chacune d'elles.

6.1 Résumé des expériences

Nous avons effectué plusieurs expériences afin de tester notre système. Ces expériences sont de plus en plus complètes. La première expérience (section 6.3) vise à se détacher de la partie « environnement ambiant » afin d'éviter les problèmes de fausses détections, de latence, de communication entre modules, etc. Elle écarte également les deux algorithmes d'apprentissage du modèle de l'environnement (l'apprentissage supervisé du modèle de transition – algorithme 5, et du modèle de renforcement – algorithme 6). Ceci nous permettra de ne tester que l'algorithme d'apprentissage par renforcement (algorithme 3). Pour ceci, nous avons utilisé des modèles écrits à la main, et un simulateur de l'environnement, décrit section 6.2. La deuxième expérience (section 6.4) se concentre sur la partie « initialisation » de l'assistant. Comme il est expliqué section 5.8, lorsque l'assistant est mis en route pour la première fois, sa Q-table est vide (son comportement est donc totalement aléatoire) mais il possède un modèle du monde initial. Ce modèle est très incomplet mais permet à l'assistant d'initialiser sa Q-table en effectuant d'entrée des épisodes d'apprentissage par renforcement indirect (algorithme 7). La deuxième expérience vise donc à choisir les paramètres optimaux pour cette phase initiale. Enfin, la troisième expérience (section 6.5) vise à évaluer le système dans sa globalité. L'assistant débute avec la Q-table résultante de l'expérience précédente, et s'exécute dans l'environnement en interagissant avec l'utilisateur pendant une période de temps prolongée. L'assistant

apprend en tâche de fond le modèle du monde avec ses algorithmes supervisés, ce qui permet d'intégrer les éléments du monde qui n'étaient pas inclus dans le modèle initial. Il exécute également, à intervalles de temps réguliers, l'algorithme d'apprentissage hors ligne afin d'intégrer dans la Q-table ces nouvelles observations.

6.2 Le simulateur de l'environnement

Nous avons mis en place une plate-forme expérimentale pour tester nos algorithmes. Le monde est simulé. Cette plate-forme remplace tous les capteurs et effecteurs de l'environnement (décrits sections 4.5.1 et 4.5.2). Le simulateur envoie des événements de la même manière que le font les capteurs. Il reçoit les commandes et y répond comme les effecteurs le feraient.

Il est possible de fournir un scénario d'échanges prédéfini. Dans ce cas, on peut considérer que l'utilisateur est également simulé. La section 6.5.1 décrit une utilisation du simulateur qui n'intègre pas l'utilisateur.

Ces échanges suivent un scénario prédéfini.

Un scénario est une séquence d'événements, par exemple :

0. « Sofia est dans le bureau » ;
1. « Nouvel email de diffusion » ;
2. « Rappel : barbecue de l'équipe à 13h » ;
3. « Sofia quitte le bureau » ;
4. « Bob entre dans le bureau » ;
5. « Sofia entre dans le bureau » ;

et ainsi de suite.

Le simulateur reçoit un tel scénario en entrée et envoie les événements correspondants de façon séquentielle. De plus, il reçoit les actions prises par l'assistant et renvoie des récompenses à celui-ci. Ces récompenses sont prédéfinies dans le scénario et l'assistant les traite exactement de la même façon que les récompenses réelles de l'utilisateur.

Le scénario prédéfinit à chaque étape l'état souhaité et la récompense à envoyer si cet état est effectivement atteint. Dans le cas contraire, nous diminuons la récompense en fonction de la distance entre l'état réellement atteint et l'état souhaité (se référer à la section 6.2.1). Pour cela, le simulateur a accès à l'état courant de l'assistant.

Ce système permet de tester facilement et automatiquement les algorithmes d'apprentissage. Les scénarios déterministes permettent de faire des expérimentations simplement car on peut tester plusieurs hypothèses dans les mêmes conditions.

6.2.1 Distance entre états

La distance entre deux états s_1 et s_2 est calculée comme le nombre de pas nécessaires pour obtenir s_2 en partant de s_1, c'est-à-dire le nombre de transitions entre s_1 et s_2 dans le graphe défini par le modèle de transition. L'algorithme de calcul de la distance est un algorithme récursif de recherche en largeur dans l'arbre formé par l'état de départ s_1 et ses successeurs. Cet arbre est représenté

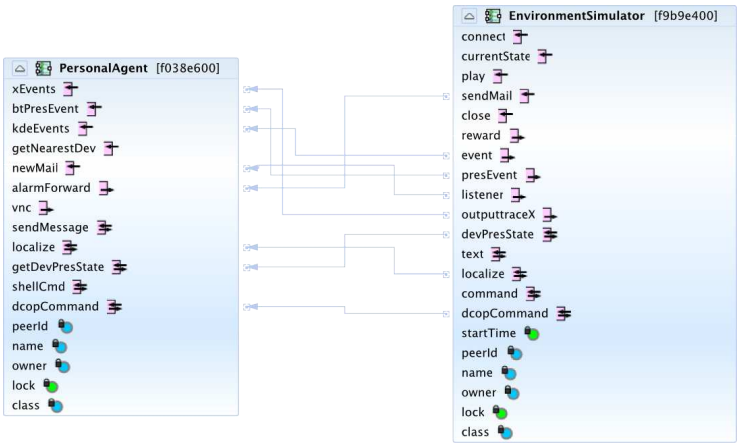

FIG. 6.1 – Le service simulant l'environnement connecté à l'assistant personnel.

sur la figure 6.2. Dans cet exemple, l'état recherché s_2 est trouvé parmi les états successeurs des successeurs de s_1, la distance entre s_1 et s_2 vaut donc 2.

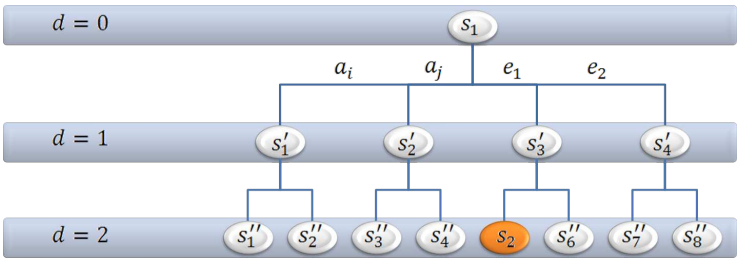

FIG. 6.2 – Exemple d'arbre produit par le modèle de transition. Dans cet exemple, $d(s_1, s_2) = 2$.

La recherche se fait sur un maximum de six étages dans l'arbre, ce chiffre étant choisi en raison du temps de calcul. Les états successeurs d'un état s sont obtenus en effectuant toutes les transitions possibles. Les transitions peuvent se faire par suite d'une action de l'assistant ou bien d'un événement de l'environnement (ceci est expliqué en détail section 5.6.5). Dans un état donné s, tous les événements peuvent survenir car ils sont indépendants de l'état de l'assistant. Par contre, l'ensemble d'actions possibles dans un état dépend de cet état. Lorsque le modèle de transition est incomplet pour un couple (s, a) ou (s, e), l'état suivant renvoyé est l'état de départ s ; dans ce cas la transition est ignorée pour le calcul de la distance. La distance entre deux mêmes états dépend donc de la connaissance actuelle qu'a l'assistant du monde.

6.3 Expérience n°1 : environnement simulé, modèle de l'environnement connu

Pour faire un premier test de notre algorithme d'apprentissage par renforcement dans un environnement ambiant, nous avons réduit le problème en

a) utilisant le simulateur plutôt que l'environnement réel afin de pouvoir facilement rejouer l'expérience en faisant varier des paramètres de l'algorithme ;

b) utilisant un modèle de l'environnement supposé correct, sans l'apprendre afin de mesurer l'efficacité de l'apprentissage par renforcement seul.

Nous avons exécuté un scénario minimal, composé de trois événements :

0. « Sofia est dans le bureau » ;
1. « Sofia quitte le bureau » ;
2. « Sofia entre dans le bureau ».

À chaque pas, neuf actions sont possibles. Elles concernent l'économiseur d'écran et la musique :

	Économiseur d'écran			Musique		
Action	Verrouiller	Déverrouiller	Rien	Pause	Jouer	Rien
1	×			×		
2	×				×	
3	×					×
4		×		×		
5		×			×	
6		×				×
7			×	×		
8			×		×	
9			×			×

TAB. 6.1 – Les neuf actions possibles lorsqu'un événement d'entrée ou sortie de personnes est détecté.

Le comportement que nous souhaitons obtenir de l'assistant est le suivant : lorsque l'utilisateur quitte le bureau, la musique se met en pause et l'écran est verrouillé. Lorsque l'utilisateur entre dans son bureau, à l'inverse, la musique rejoue et l'écran est déverrouillé.

La règle pour donner les récompenses est la suivante : lorsque l'action est bonne pour les deux paramètres (l'écran et la musique), le renforcement est très bon (50). Lorsque les deux paramètres sont mauvais, on donne un renforcement fortement négatif (-50). Enfin, lorsque seule l'une des deux actions est mauvaise, le renforcement donné est moins négatif (-25). Le modèle de renforcement fourni est fait « à la main » pour correspondre à ces valeurs. Le modèle de transitions est également fait « à la main » et englobe les états rencontrés lors de ce simple scénario.

Au départ de l'expérience, la Q-table est vide et l'on commence par exécuter un épisode de Q-Learning de 20 itérations. Après cet épisode exécuté pour initialiser le comportement, nous exécuterons un épisode plus court, de 10 itérations,

chaque minute. Chaque épisode démarre dans le même état initial par défaut et rejoue des événements choisis au hasard dans la base de données.

Afin d'évaluer la politique apprise, le simulateur joue en parallèle et en boucle le scénario décrit ci-dessus. Après 50 épisodes, nous modifions le taux d'exploration de notre politique ϵ-*gloutonne* de 0.5 à 0.1. Ceci signifie que 90% des fois, c'est la meilleure action qui est choisie. Pour mesurer l'évolution de la somme des renforcements donnés, nous avons rejoué ce scénario près de 800 fois et nous avons calculé cette somme des renforcements reçus pour chaque instance du scénario.

Remarque : L'exécution d'un scénario n'a aucune influence sur l'apprentissage (elle en aurait sur l'apprentissage du modèle du monde, mais celui-ci n'est pas effectué dans cette expérience). Elle permet simplement de visualiser l'évolution de la politique apprise. Cette expérience ne veut donc en aucun cas dire que l'utilisateur devrait entrer et sortir de son bureau 800 fois !

Les résultats de cette expérience sont présentés dans la section 7.2.

6.4 Expérience n°2 : autour de l'apprentissage initial

Cette expérience a pour but de comparer les différents paramètres pouvant varier dans la phase initiale, au premier démarrage du système. Cette phase consiste à transformer le modèle de l'environnement initial en une Q-table initiale et donc un comportement initial. En effet, il est plus facile de spécifier des transitions et des exemples de récompenses que des Q-valeurs (la section 5.8 traite de cet aspect). Afin d'initialiser la Q-table de manière à refléter ce modèle de l'environnement initial, nous exécutons des épisodes hors-ligne de Q-Learning. On intègre ainsi à la Q-table la connaissance que nous avons du monde.

Les paramètres que nous pouvons faire varier sont les suivants :

1. Le nombre d'épisodes (`NB_EP`) ;
2. Le nombre d'itérations de chaque épisode (`NB_ITER`) ;
3. L'état initial de chaque épisode (`INIT_STATE`) avec les trois possibilités :
   - L'état par défaut (tous les arguments ont des valeurs nulles `<null>`) (`DEF`) ;
   - Un état dont toutes les valeurs sont tirées au hasard (`RND`) ;
   - Un état tiré au hasard parmi tous les états déjà rencontrés (pour cette option il faut donc attendre que le système ait commencé à tourner et à enregistrer dans la base de données les états rencontrés) (`RND_DB`).
4. Le choix de l'événement à chaque pas de chaque épisode (`EVENT_GEN`), deux possibilités :
   - Un événement généré aléatoirement (`RND`) ;
   - Un événement choisi aléatoirement parmi tous les événements enregistrés dans la base de données (de même, ceci ne peut être fait qu'après le démarrage du système) (`RND_DB`).

Nous avons donc exécuté des tests pour déterminer les meilleurs choix parmi les valeurs suivantes des paramètres définis ci-dessus (tableau 6.2).

Remarque : Il est intéressant de remarquer pourquoi nous choisissons d'effectuer plusieurs épisodes pour cette phase initiale, au lieu d'un seul plus long,

Paramètre	Valeurs possibles
NB_EP	de 1 à 100
INIT_STATE	DEF, RND, RND_DB
NB_ITER	10, 25, 50, 100
EVENT_GEN	RND, RND_DB

TAB. 6.2 – Paramètres variant dans l'expérience n°2 et leurs valeurs.

comme nous avons d'abord pensé le faire (section 6.3). D'abord, plusieurs épisodes représentent plusieurs chemins dans le graphe formé par le modèle de transition. Effectuer plusieurs chemins apporte une plus grande probabilité d'explorer davantage l'espace d'états-actions, d'autant plus si chaque épisode démarre dans un état aléatoire. Ensuite, un épisode a toujours une possibilité de se bloquer. Comme il est expliqué section 5.10, si le modèle de transition ne peut pas fournir l'état suivant lors d'une itération, alors l'itération est abandonnée et un nouvel événement est choisi[1]. Ainsi, les épisodes dépendent du modèle de l'environnement, de la connaissance actuelle qu'a l'assistant du monde. Dans le cas de cette initialisation du comportement, le modèle du monde est particulièrement réduit. Il y a, par conséquent, une probabilité forte qu'il existe des états terminaux dans lesquels le modèle ne connaît l'issue d'aucune action ni d'aucun événement. Si nous effectuons un seul épisode et que nous rencontrons un tel état, le résultat de l'apprentissage risque d'être particulièrement faible. Faire plusieurs épisodes permet de se débloquer d'une telle situation.

Les résultats de cette expérience sont présentés dans la section 7.3.

6.5 Expérience n°3 : Intégration des interactions avec l'utilisateur et de l'apprentissage par renforcement et supervisé

Enfin, nous avons lancé ensemble toutes les parties de notre assistant. Toutefois, nous avons subdivisé cette expérience car, dans un premier temps, nous avons encore utilisé le simulateur du monde, mais d'une manière différente de la première expérience.

6.5.1 « Le tableau de bord »

Le tableau de bord est une interface graphique intégrée au simulateur de l'environnement et qui propose des boutons pour envoyer à l'assistant tous les types événements définis dans l'environnement. Cette interface est montrée figure 6.3. Ceci se rapproche beaucoup plus du cas réel que l'utilisation précédente du simulateur (avec des scénarios) et permet simplement d'éviter les problèmes liés

[1] *Remarque* : ce n'est pas le cas avec le modèle de récompense. En effet, lorsque celui-ci n'a pas d'informations à fournir, il retourne un renforcement de zéro. Ceci n'est pas une raison pour abandonner l'itération car, dans le cas réel, si l'utilisateur ne fournit pas de renforcement, l'algorithme s'effectue avec un renforcement nul (alors que, dans le cas réel, il y a toujours un état suivant).

aux capteurs réels, mais aussi de simplifier la vie du développeur qui souhaite générer des événements pour tester le système.

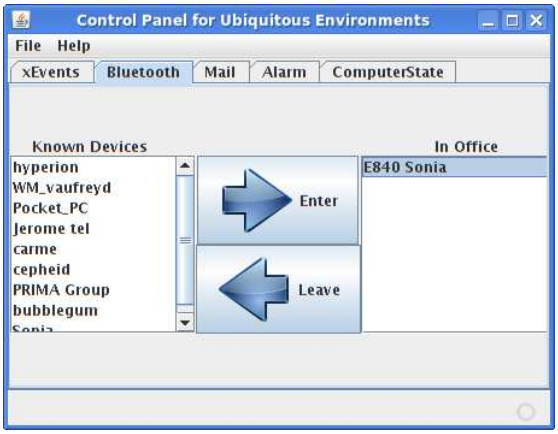

FIG. 6.3 – Une capture d'écran de l'interface « tableau de bord ». L'onglet sélectionné ici permet d'envoyer des événements d'entrée/sortie de dispositifs Bluetooth enregistrés dans la base de données.

6.5.2 Expérience

Pour cette expérience, nous sommes partis de la Q-table gagnante de l'expérience précédente (section 6.4), qui représente le comportement initial de notre assistant, obtenu en « traduisant » le modèle du monde par défaut en comportement.

L'assistant interagit avec le simulateur de l'environnement, contrôlé par l'expérimentateur. Tous les six pas (événements de l'environnement ou actions de l'assistant), l'apprentissage supervisé des modèles de transition et de récompense se déclenche afin d'intégrer les dernières informations enregistrées. Les épisodes d'apprentissage par renforcement hors-ligne s'exécutent en tâche de fond un par un, avec une minute d'intervalle entre deux épisodes. Ces intervalles d'une minute et de six pas ont été choisis afin d'accélérer l'expérience, puisque l'expérimentateur est là pour générer des événements fréquents. Dans « la vie réelle », les événements seront moins fréquents et l'intervalle pourra être augmenté progressivement. Après une première phase d'adaptation active à l'utilisateur, nous pourrons exécuter un épisode[2] par jour par exemple. En effet, l'utilisateur pourra bien sûr modifier ses préférences, pour cette raison nous n'arrêterons jamais définitivement l'apprentissage, mais ces modifications seront peu fréquentes et à grande échelle.

Nous avons laissé « tourner » l'assistant dans ces conditions pendant plusieurs jours, en passant un peu de temps à lui fournir de nouveaux événements et des renforcements. Certains de ces renforcements étaient dans la continuité de ce qui était déjà présent dans le modèle initial, et certains étaient nouveaux car ils répondaient à des événements non modélisés par défaut. Les résultats de cette expérience seront présentés section 7.4.

[2]Une phase d'apprentissage par renforcement qui intégrera les entrées de l'utilisateur au comportement.

6.6 Validation croisée de l'algorithme d'apprentissage supervisé de la fonction de transition

L'algorithme d'apprentissage supervisé du modèle de transition est décrit section 5.9.1. Nous avons mis en avant le fait que cet algorithme permet une généralisation des exemples sur lesquels il apprend le modèle. Ceci permet à l'apprentissage par renforcement hors ligne (algorithme 7 section 5.10) d'explorer une plus grande partie de l'espace d'états-actions, et ainsi d'avoir une estimation du bon comportement dans un état qui n'a encore jamais été observé.

Afin de vérifier cette capacité de généralisation, nous avons effectué une validation croisée à 10 plis *(10-fold cross-validation)*. L'ensemble des exemples de transition $Ex = \{\{s, o, s'\}\}$ de la base de données a été aléatoirement divisé en un ensemble de 10 sous-parties disjointes formant une partition : $Ex = \{ex_i, i \in [0, 9]\}$. L'algorithme de validation croisée est résumé par l'algorithme 8 ci-dessous.

Algorithme 8 : Validation croisée de l'apprentissage supervisé du modèle de transition.

Entrée : Une partition de l'ensemble d'exemples $Ex = \{ex_i, i \in [0, 9]\}$
Sortie : Une matrice de confusion
Pour chaque *partition* ex_k, $k \in [0, 9]$ **faire**
    Apprendre le modèle de transition sur les exemples des partitions ex_j, $j \in [0, 9] \setminus k$;
    Tester le modèle appris sur les exemples de la partition ex_k;

Cet algorithme permet de tester le modèle appris sur des exemples ne faisant pas partie de l'ensemble d'apprentissage et ainsi d'estimer sa capacité de généralisation. De plus, la partition étant aléatoire et l'apprentissage étant effectué dix fois sur différents ensembles d'exemples, ce test est indépendant des données et du choix de la partition. Le résultat de cette validation croisée est présenté dans la section 7.5.

Chapitre 7

Résultats et interprétation

Dans ce chapitre, nous allons décrire et commenter les résultats de chacune des expériences décrites dans le chapitre précédent (chapitre 6). Avant cela, dans la section 7.1, nous allons préciser les critères importants pour évaluer notre assistant et le système de mesure que nous avons mis en place.

7.1 Mesure de qualité d'un résultat

Dans un système d'apprentissage par renforcement, le résultat de l'apprentissage est une politique π qui, en se basant sur une représentation de la fonction de valeur d'action, donne un comportement au système. Notre premier critère d'évaluation est donc le comportement obtenu, et, plus précisément, à quel point celui-ci se rapproche du comportement désiré. En effet, le comportement de l'assistant a un impact direct sur l'utilisateur. D'autre part, ce qui a également un impact sur l'utilisateur est le temps nécessaire au système pour obtenir le comportement désiré. Nous devons donc également mesurer la vitesse de l'apprentissage.

Par conséquent, nous allons définir une *note* numérique calculée pour une Q-table donnée. Cette note permettra d'évaluer le premier critère – la désirabilité du comportement à un instant donné. Le deuxième critère – la vitesse de l'apprentissage – pourra être mesuré en comparant les Q-tables obtenues lors d'épisodes hors-ligne consécutifs d'apprentissage par renforcement.

Si l'on se place du point de vue de l'utilisateur et en laissant de côté l'exploration dans le choix de l'action, alors le critère important est le nombre de situations dans lesquelles l'assistant va agir de manière satisfaisante pour l'utilisateur. Dit autrement, il s'agit du nombre d'états pour lesquels la meilleure action de la Q-table est effectivement la meilleure action selon l'utilisateur. Dans la suite, nous allons appeler ce nombre, le nombre de Q-valeurs (subjectivement) correctes, `nbCorrect`. L'évaluation du caractère correct ou incorrect d'une Q-valeur est faite par l'expérimentateur qui a spécifié les modèles initiaux de transition et de récompense. En effet, la Q-table résultante doit correspondre à ces modèles fournis, il faut donc que l'évaluation des Q-valeurs suive la même logique que le modèle de l'environnement (le modèle initial fourni par défaut, puis les renforcements donnés en cours d'exécution et qui sont intégrés dans le modèle par le biais de l'apprentissage supervisé).

Ce nombre de Q-valeurs correctes est donc la partie majeure de la note finale d'une Q-table. Mais nous allons introduire deux autres critères ayant des poids bien plus faibles afin de départager des Q-tables dont les nombres de Q-valeurs correctes sont proches. Le nombre total de Q-valeurs de la table reflète la couverture de l'espace d'états-actions. Plus cette couverture est grande, plus il y a d'états dans lesquels on a une idée du comportement à avoir et donc l'action choisie ne sera pas totalement aléatoire, ce qui est plus satisfaisant pour l'utilisateur. Ce nombre sera noté `nbTotal`. Parmi ces couples état-action visités, certains ont des Q-valeurs nulles car le modèle n'a pas renvoyé de récompense et, par conséquent, la Q-valeur n'a pas pu être modifiée (et elle n'a pas non plus été modifiée par propagation). Le pourcentage de Q-valeurs non nulles dans la Q-table – `pNonNul` – est donc un critère. Il reflète effectivement le nombre d'états dans lesquels l'assistant a un comportement approximatif alors que le nombre total d'entrée montre le potentiel d'exploration. La note finale d'une Q-table est donc obtenue selon la formule :

$$\texttt{note} = \frac{1}{13}\left(10 \times \texttt{nbCorrect} + 2 \times \texttt{pNonNul} + \texttt{nbTotal}\right) \tag{7.1}$$

7.2 Résultats de l'expérience n°1 : environnement simulé, modèle de l'environnement connu

Cette première expérience, décrite section 6.3, était minimale et simplement destinée à tester l'apprentissage par renforcement pur (le modèle du monde était fourni et « idéal »), les événements et les renforcements étaient fournis par le simulateur de l'environnement, selon un scénario prédéfini.

La figure 7.1 montre les renforcements donnés au cours des scénarios et leur régression linéaire. Les récompenses ont une grande variation, mais la régression linéaire montre qu'ils ont une progression positive. Autour de la $500^{\text{ème}}$ exécution du scénario, nous avons diminué le taux d'exploration. Nous pouvons observer sur la courbe qu'à partir de ce moment-là, la plupart des récompenses données sont positives.

Ces résultats préliminaires nous ont, dans un premier temps, encouragés car l'apprentissage fonctionnait. Dans un deuxième temps ils nous ont montré la nécessité d'augmenter le nombre d'itérations par épisode. Nous avons intuitivement pensé que le premier épisode devrait avoir au moins quatre ou cinq cent itérations et que les épisodes suivants devraient tous avoir environ cent itérations. L'expérience suivante (décrite section 6.4), a été destinée à déterminer les paramètres optimaux pour l'épisode initial. Nous avons gardé ce même nombre d'itérations optimal pour les épisodes suivants. Par contre, nous avons décidé de ne pas exécuter un seul épisode initial très long, mais plusieurs plus courts afin de les faire démarrer dans différents états initiaux et ainsi couvrir une plus grande partie de l'espace d'états-actions (on effectue ainsi une sorte d'exploration « masquée »). Ceci est décrit plus en détails dans la section 6.4.

Il est également intéressant de constater la convergence des Q-valeurs. Ceci est illustré figure 7.2. Dans cette figure, l'axe des abscisses représente les pas du Q-Learning. L'axe des ordonnées représente la valeur de la mise à jour moyenne de la Q-table. À chaque étape, plusieurs Q-valeurs sont modifiées afin de propager

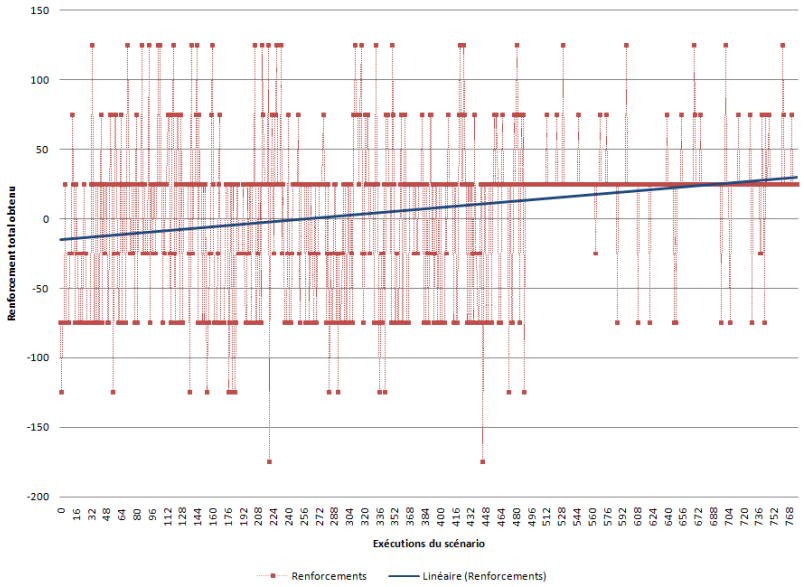

FIG. 7.1 – L'évolution du renforcement total donné pour chaque scénario, et sa régression linéaire.

un changement. Les valeurs y sont les moyennes des valeurs absolues des différences entre les anciennes et les nouvelles Q-valeurs : $y = \frac{1}{n_x} \sum_{i=1}^{n_x} (|Q_i^x - Q_i^{x-1}|)$, où $\{Q_i^x,\ i \in [1;\ n_x]\}$ est l'ensemble des Q-valeurs modifiées lors du pas x. Comme il est clair sur la figure 7.2, cette différence diminue radicalement, ce qui signifie que les Q-valeurs sont de moins en moins modifiées. Cette moyenne peut être utilisée pour mettre fin à un épisode avant que le nombre d'itérations demandé ne soit atteint, car si cette valeur est trop faible, cela signifie que la Q-table n'est plus mise à jour (le changement a été intégré dans la Q-table). Ainsi, l'épisode n'est plus utile. Nous attendrons que le modèle de l'environnement soit modifié, ou bien que suffisamment de nouveaux événements soient ajoutés à la base de données pour commencer un nouvel épisode.

7.3 Résultats de l'expérience n°2 : autour de l'apprentissage initial

Nous avons exécuté différents tests afin de trouver les meilleures valeurs des différents paramètres regroupés dans le tableau 6.2. Nous avons calculé les notes de chacune des Q-tables résultantes selon la méthode décrite dans la section 7.1.

Le premier constat a été le fait que les événements aléatoires lors d'épisodes (`EVENT_GEN=RND`) ne font pas décoller l'apprentissage. Les notes obtenues pour les épisodes effectués avec ce paramètre étaient toujours nulles. En effet, le modèle du monde initial que nous avons utilisé était trop restreint et n'a jamais réussi à trouver l'état successeur sur un état et un événement généré aléatoirement. Lorsque le modèle ne connaît pas une transition, nous abandonnons l'itération en cours. Dans ce cas, toutes les itérations ont ainsi été abandonnées

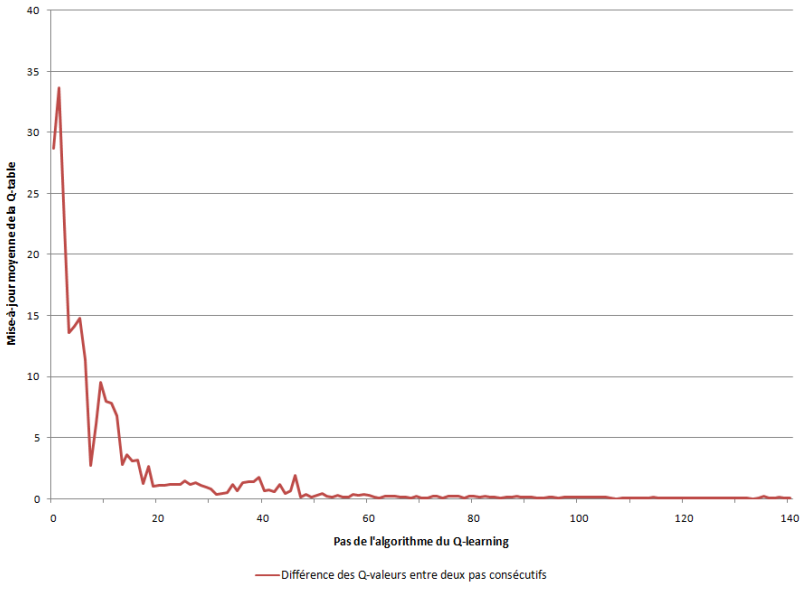

FIG. 7.2 – La mise à jour moyenne des Q-valeurs à chaque pas de l'algorithme du Q-Learning (algorithme 1).

et l'apprentissage n'a rien fait. Les événements aléatoires sont toutefois envisageables pour les épisodes ultérieurs, lorsque le modèle du monde sera plus général. Dans la suite de cette section, les épisodes sont effectués en choisissant les événements aléatoirement dans la base de données.

Nous avons également souhaité comparer l'influence des différents paramètres, en commençant par l'état initial. Les courbes de la figure 7.3 montrent les notes obtenues en exécutant différents nombres d'épisodes (axe des abscisses), chacun de 100 itérations et chacun démarrant dans l'état initial par défaut (courbe rouge), un état aléatoire (courbe verte), ou bien un état aléatoire choisi parmi les états déjà rencontrés et enregistrés dans la base de données. Le constat fait à partir de cette figure est que l'état initial tiré aléatoirement parmi les états déjà rencontrés permet un apprentissage nettement plus efficace que les autres techniques. L'état initial par défaut peut être utilisé si une base de données d'états n'est pas disponible, l'état initial par défaut étant la moins bonne des solutions. Ce résultat s'explique également par le fait qu'en démarrant dans un état existant, l'épisode a plus de chances de démarrer dans un état « géré » par le modèle de transition (d'autant plus que les événements sont également choisis parmi ceux déjà rencontrés). Bien que ce modèle soit encore minimal, il a tout de même été construit dans le but de correspondre à des éléments observables, il contient des états et événements couramment rencontrés. En résumé, le modèle étant bien adapté à ces états, l'apprentissage est plus efficace.

Les résultats suivants ont, par conséquent, été effectués avec le choix des états aléatoires parmi ceux enregistrés en tant qu'états initiaux des épisodes.

Enfin, nous avons voulu mesurer l'influence du nombre d'itérations par épisode. Ce test est résumé par la figure 7.4.

Ce résultat montre que, comme attendu, la note est d'autant meilleure que le

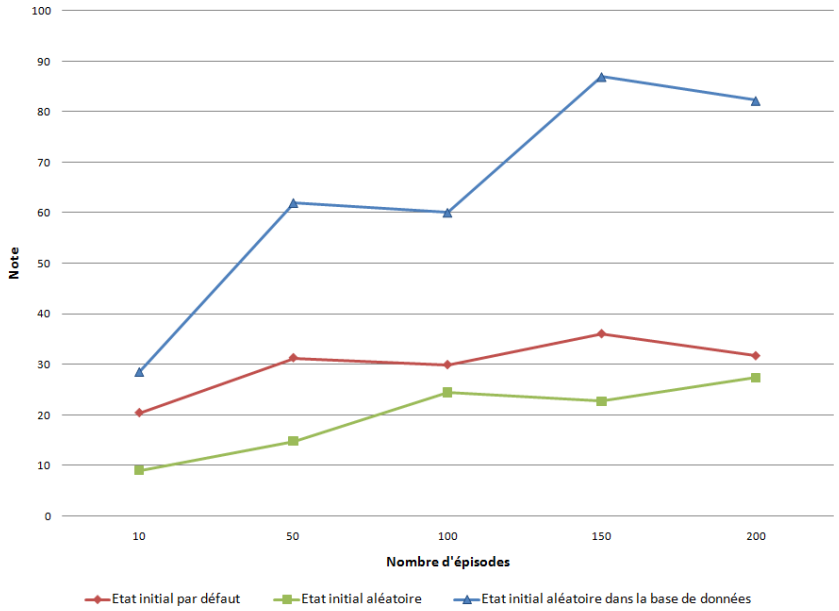

FIG. 7.3 – Influence de la façon de choisir l'état initial avec 100 itérations par épisode et événements tirés au hasard dans la base de données.

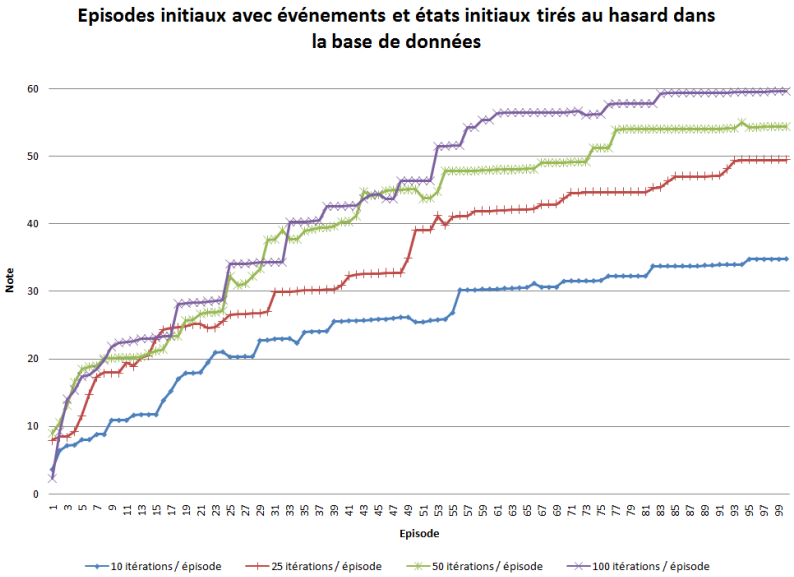

FIG. 7.4 – Influence du nombre d'itérations par épisode avec état initial et événements tirés au hasard dans la base de données.

nombre d'itérations par épisode est important. En effet, faire un grand nombre d'épisodes d'un grand nombre d'itérations chacun permet d'explorer mieux l'espace d'états-actions, par conséquent d'essayer plus d'actions et avoir plus de

chances de tomber sur la bonne. Il est également intéressant de remarquer que l'augmentation de la note n'est pas proportionnelle à l'augmentation du nombre d'itérations, ni du nombre d'épisodes. En effet, multiplier le nombre d'épisodes par 10 ne multiplie pas la note par 10, de même pour le nombre d'itérations par épisode. Par exemple, les configurations suivantes permettent toutes d'obtenir la note 20 :

- 22 épisodes de 10 itérations, soit 220 itérations au total ;
- 14 épisodes de 25 itérations, soit 350 itérations au total ;
- 9 épisodes de 50 itérations, soit 450 itérations au total ;
- 9 épisodes de 100 itérations, soit 900 itérations au total ;

En moyenne, toutes les itérations s'exécutent en un temps équivalent et le nombre d'itérations abandonnées à cause d'une lacune dans le modèle est le même quel que soit le nombre d'itérations (car ce phénomène est indépendant de l'apprentissage). Ces données montrent qu'il faut approximativement quatre fois plus d'itérations pour atteindre la note 20 avec des épisodes de 100 itérations chacun qu'avec des épisodes de 10 itérations chacun. Il est donc bien plus efficace de faire le choix des 10 itérations par épisode pour atteindre la note de 20 le plus rapidement possible.

Par contre, les épisodes de 100 itérations permettent d'atteindre la note la plus haute au final. Les quatre courbes de la figure 7.4 se stabilisent autour de l'épisode 80 et il est logique d'inférer qu'elles ne se croiseraient pas si l'on continuait l'expérience. Nous en déduisons que des épisodes longs permettent un apprentissage d'une meilleure qualité, mais plus lent.

Le choix du paramètre final devrait équilibrer la qualité du résultat avec le temps d'exécution. Il faut garder à l'esprit qu'il ne s'agit ici que d'épisodes servant à initialiser le comportement et que l'apprentissage se poursuivra activement après cette phase. Il n'est donc pas obligatoire de choisir la configuration avec le meilleur résultat possible, mais un bon compromis par rapport au nombre total d'itérations à effectuer pour l'atteindre. Le choix le plus efficace est celui de 10 itérations par épisode, mais la note maximale pouvant être atteinte (la qualité de l'apprentissage) semble trop faible par rapport aux autres choix. Effectuer, par exemple, 50 épisodes de 25 itérations semble raisonnable.

7.4 Résultats de l'expérience n°3 : Intégration des interactions avec l'utilisateur et de l'apprentissage par renforcement et supervisé

Nous allons maintenant présenter le résultat de la troisième expérience décrite section 6.5, qui intègre apprentissage par renforcement et supervisé, et qui se base sur des événements générés par l'expérimentateur au travers d'une interface décrite section 6.5.1. Comme nous l'avons précisé dans la section 6.5, nous avons exécuté des épisodes d'apprentissage par renforcement séparés par un intervalle d'une minute. Au total nous avons exécuté près de 120 épisodes. Entre temps, nous interagissions avec l'assistant en lui fournissant de nouveaux événements et de nouveaux renforcements. Chaque épisode modifie la Q-table. Nous avons calculé la note de chacune de ces tables : la figure 7.5 regroupe ces résultats.

Comme nous pouvons le voir sur la figure 7.5, les notes des épisodes consé-

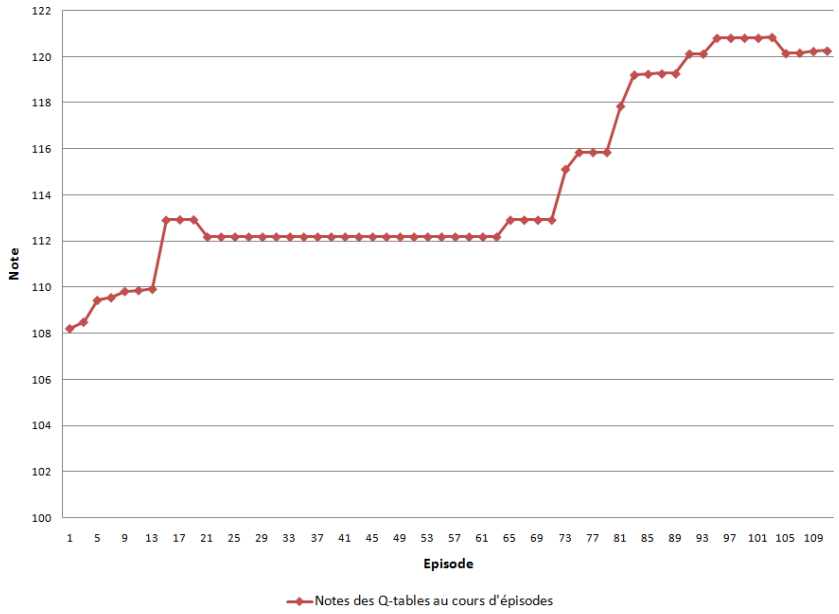

FIG. 7.5 – Notes des Q-tables produites par chaque épisode d'apprentissage par renforcement exécuté en parallèle d'interactions avec l'assistant.

cutifs sont croissantes. Ceci signifie que le comportement appris est de plus en plus adéquat aux préférences de l'utilisateur. Les deux zones dans lesquelles la courbe est fortement croissante (correspondant approximativement aux intervalles $x \in [0, 20]$ et $x \in [70, 110]$) correspondent aux périodes où l'expérimentateur générait de nouveaux événements et donnait des renforcements, fournissant ainsi à l'assistant beaucoup de matériel à intégrer dans la Q-table. La zone dans laquelle la courbe est plate (pour les abscisses approximatives $x \in [20, 70]$) correspond à un moment où l'expérimentateur s'est absenté et où il n'y avait donc aucun événement ni renforcement. Ces épisodes n'ont pas fait évoluer l'apprentissage car la Q-table avait alors intégré toutes les informations disponibles à cet instant.

La courbe montrée figure 7.6 représente approximativement la même expérience que la figure 7.5. L'expérience démarre après la phase d'apprentissage initial, et intègre tous les nouveaux événements fournis par l'expérimentateur. Autour de l'épisode 90, l'expérimentateur a décidé de « changer d'avis » en donnant désormais un renforcement contradictoire à la logique suivie précédemment. On constate sur la courbe que la note diminue d'une manière significative (la figure 7.7 montre un agrandissement de cette zone d'intérêt). Cette baisse correspond aux états dans lesquels la meilleurs action ne correspond plus à l'action désirée par l'expérimentateur, alors qu'elle correspondait jusqu'à présent. Puis, la note remonte progressivement, alors que les changements sont appris et intégrés d'abord au modèle de renforcement, puis au comportement.

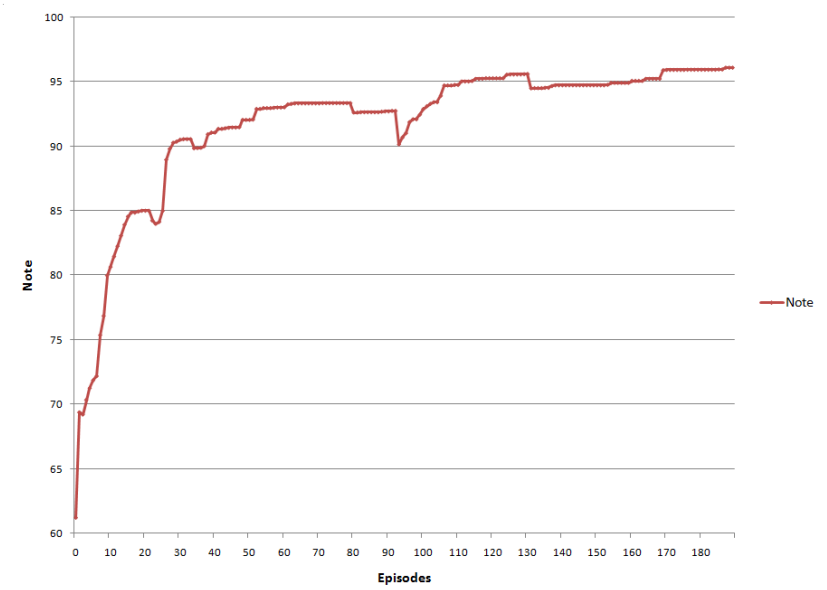

FIG. 7.6 – Notes des Q-tables produites par chaque épisode d'apprentissage par renforcement, avec une baisse soudaine correspondant à un changement d'avis de l'utilisateur, puis la remontée correspondant à l'adaptation du système.

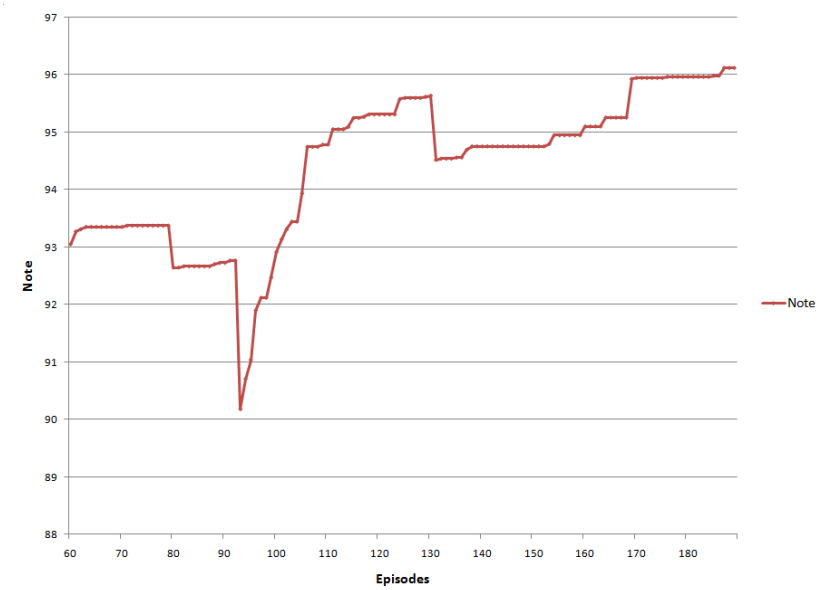

FIG. 7.7 – Un agrandissement de la courbe figure 7.6 sur la zone de la chute de la note, puis de la remontée.

7.5 Résultat de la validation croisée de l'algorithme d'apprentissage supervisé de la fonction de transition

La section 6.6 présente une validation croisée que nous avons effectuée pour vérifier la capacité de généralisation de l'algorithme d'apprentissage supervisé du modèle de transition présenté section 5.9.1 (algorithme 5). Nous avons utilisé les 3352 exemples disponibles dans la base de données. Ceux-ci ont été séparés en 10 sous-ensembles disjoints aléatoires. À chaque étape de la validation croisée, l'ensemble d'apprentissage était donc en moyenne composé de 3016 exemples, et l'ensemble de test était formé des 336 (en moyenne) exemples restants.

Le résultat de la validation croisée est une matrice de confusion dont la construction est expliquée par le tableau 7.1. Les lignes et les colonnes représentent tous les états rencontrés dans les exemples. Le test du modèle appris est fait sur chaque exemple de la partition k de la manière suivante. L'exemple nous fournit un triplet $\{s, o, s'\}$: un état s et l'état suivant s' qui a été observé lorsque l'occurrence o est survenue dans s. Le test donne s et o en entrée au modèle, et récupère sa sortie : s'_m, l'état suivant estimé. On incrémente la case de la matrice dont la ligne est celle de l'état réellement obtenu dans l'exemple (s') et dont la colonne est celle de l'état estimé par le modèle (s'_m). Dans l'exemple hypothétique du tableau 7.1, lorsque le s' réel était s_i, le modèle a également trouvé s_i dix fois, mais trois fois, il a trouvé s_j au lieu de s_i.

Vu le grand nombre de nos états, nous avons représenté ce tableau sous forme d'une image, donnée figure 7.8. Chaque pixel de cette image correspond à une case de la matrice de confusion. Les valeurs de 0 dans la matrice correspondent aux pixels blancs dans l'image. La couleur s'assombrit lorsque le nombre de « correspondances » augmente. Nous pouvons constater que seuls quelques pixels en dehors de la diagonale ne sont pas blancs. Ceci signifie que le modèle renvoie rarement un état suivant différent de l'état suivant réellement rencontré ($s'_m \neq s'$). Dans la majorité des cas, $s'_m = s'$. Étant donné que les exemples de test n'ont pas été appris, le modèle possède bien une capacité de généralisation.

État estimé / État réel	s_i	s_j	s_k
s_i	10	3	0
s_j	0	15	1
s_k	2	1	12

Tab. 7.1 – Représentation de la matrice de confusion.

7.6 Conclusion

Les expériences menées montrent la capacité de notre système à apprendre un comportement à partir d'interactions avec l'environnement et l'utilisateur limitées en nombre. La croissance des notes des Q-tables successives démontre la réussite de l'apprentissage. Elle signifie que dans plus en plus d'états l'assistant

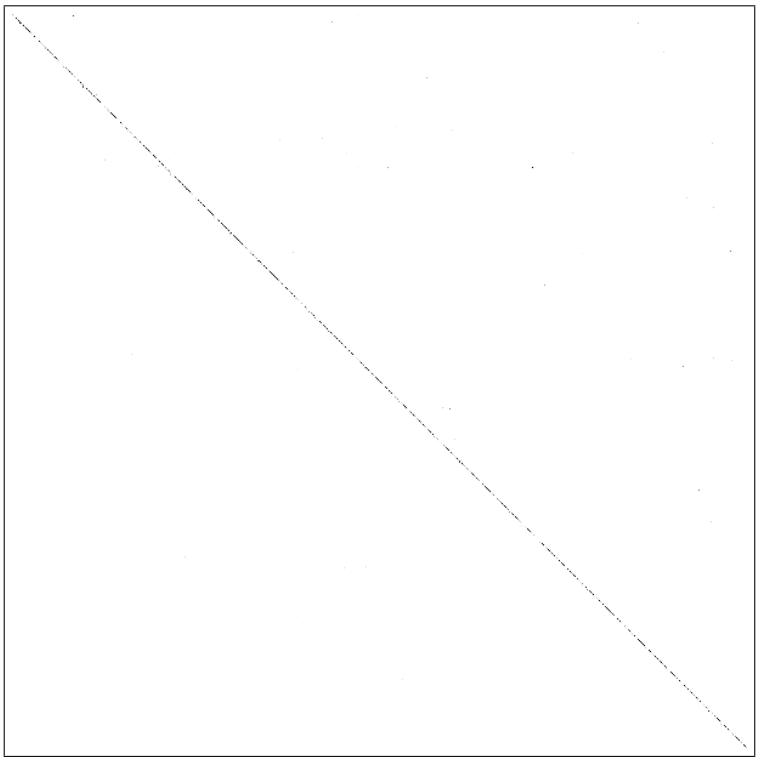

FIG. 7.8 – La matrice de confusion produite par l'algorithme 8 de validation croisée.

choisira la « bonne » action, celle que souhaite voir exécutée l'expérimentateur et donc, à terme, l'utilisateur. Cette croissance de la note prouve également l'exactitude de notre modèle de l'environnement. En effet, l'apprentissage par renforcement s'effectue uniquement sur le modèle du monde, il ne peut, par conséquent, pas réussir si le modèle est faux. Par ailleurs, la validation croisée décrite section 7.5 a montré la capacité de généralisation du modèle de transition.

Chapitre 8

Conclusion et perspectives

8.1 Récapitulatif

Un environnement équipé d'un système ambiant devient intelligent lorsqu'il peut percevoir le contexte de l'utilisateur et agir en fonction de sa situation actuelle, de manière appropriée. En détectant le contexte, le système ne parcourt que la moitié du chemin car il lui reste alors le choix de l'action à exécuter. Nous avons mis en avant des arguments en faveur de l'acquisition automatique des associations optimales pour un utilisateur donné entre les situations du contexte et les actions du système. Ces associations ne peuvent être entièrement prédéfinies par un expert car le système serait alors trop peu adapté à l'utilisateur et ce dernier aurait des difficultés à l'appréhender et le comprendre. Il ne ferait, par conséquent, pas confiance au système. Or, nous pensons que la confiance de l'utilisateur est une condition nécessaire à l'acceptation du système par l'utilisateur, et donc du succès de ce système. L'utilisateur ferait naturellement confiance à un système qu'il aurait lui-même spécifié, mais ce serait une tâche trop fastidieuse vue la complexité et la richesse de l'environnement réel. Nous ne pouvons éventuellement demander à l'utilisateur de spécifier qu'une partie du système. Dans ces deux cas, le système ne pourrait que difficilement s'adapter aux évolutions du monde et de l'utilisateur. Afin de le maintenir à jour, l'intervention de l'expert ou de l'utilisateur serait nécessaire.

L'apprentissage du modèle de contexte est donc une bonne approche car les associations sont acquises automatiquement et non pas prédéfinies par une personne. Un apprentissage à vie permet au système de s'actualiser lorsque ceci est nécessaire. Nous apprenons les préférences de l'utilisateur, par conséquent c'est lui-même qui doit entraîner le système. Cet entraînement doit alors être simple et les résultats – visibles rapidement. Pour gagner la confiance de l'utilisateur, le système doit être transparent, il ne doit pas être vu par l'utilisateur comme une boîte noire. Ceci peut être acquis par la possibilité de donner des explications sur le comportement du système. Les représentations internes utilisées doivent être humainement lisibles.

L'apprentissage par renforcement permet de remplir ces conditions et d'atteindre le but. L'entraînement est naturellement simple pour l'utilisateur puisqu'il s'agit de récompenses subjectives, positives ou négatives, reflétant la satisfaction de l'utilisateur vis-à-vis du comportement du système. La rapidité de

l'apprentissage peut être atteinte par une technique d'apprentissage par renforcement indirect. Le système apprend un modèle de l'environnement à partir d'interactions réelles et utilise ce modèle pour simuler des expériences supplémentaires et ainsi apprendre plus rapidement.

Nous avons proposé un modèle des transitions de l'environnement et un modèle des récompenses utilisateur, tous deux appris par des algorithmes d'apprentissage supervisé à partir de l'historique des transitions et récompenses de l'environnement réel. De surcroît, le modèle des transitions effectue une généralisation des exemples sur lesquels il apprend. Ainsi, il est plus vaste que la partie de l'environnement ayant réellement été explorée. Cette généralisation est possible grâce à une modélisation « gros grain » de l'environnement. Elle permet d'apprendre un comportement approximatif pour des situations n'ayant jamais encore été observées. Lorsque celles-ci seront rencontrées pour la première fois, le système n'aura pas un comportement incohérent.

De manière similaire, nous ne voulons pas que le système ait un comportement initial absurde. Malgré les avantages qu'apporte l'apprentissage du modèle de contexte, il souffre d'un inconvénient majeur : a priori il démarre à zéro. Nous comblons ce manque par une injection de connaissances initiales. En effet, l'utilisation d'une entrée minimale de la part d'un humain est un très grand avantage pour l'apprentissage. Nous exploitons des connaissances basiques, de sens commun, sur l'environnement pour fournir un modèle initial du monde et l'utiliser pour apprendre un comportement initial. Il est effectivement plus facile de fournir une description du monde initiale que directement un comportement.

Pour tester notre approche, nous avons développé un système ambiant. Celui-ci est composé de modules distribués qui sont de trois types non exclusifs : les capteurs, les effecteurs et le module central appelé assistant personnel. L'assistant reçoit des informations des capteurs et en déduit le contexte de l'utilisateur. Il utilise sa politique apprise pour choisir une action à exécuter et ainsi rendre un service à l'utilisateur. Cette action est envoyée aux effecteurs. Les composants du système doivent donc pouvoir communiquer. De plus, cette architecture doit être dynamique car les services rendus dépendent du contexte de l'utilisateur qui n'est connu qu'à la dernière minute. Ce contexte influe sur les paramètres de l'action (par exemple la pièce dans laquelle se trouve l'utilisateur ou bien l'écran le plus proche de lui) mais également sur la modalité de l'action. Par exemple, une même action « informer l'utilisateur » peut être exécutée différemment selon le contexte. Si l'utilisateur est seul, un message vocal peut lui être envoyé, s'il est dehors, il peut s'agir d'un SMS ou encore s'il est en réunion, le SMS sera choisi, mais le téléphone portable de l'utilisateur sera mis en mode silencieux. Les dispositifs portables, tels que les téléphones cellulaires et PDAs font partie de l'environnement et peuvent être utilisés comme capteurs ou effecteurs. Or, ces appareils sont transportés par les utilisateurs et peuvent donc apparaître et disparaître dynamiquement et de manière imprévisible.

Le système ambiant doit tenir compte de ces contraintes. De plus, il ne doit jamais être arrêté, et donc il doit être mis à jour à chaud. L'architecture que nous avons mise en place utilise la plate-forme dynamique à composants OSGi et l'intergiciel OMiSCID. La combinaison de ces deux technologies permet de connecter des modules et de les faire communiquer. De plus, elle permet de contrôler automatiquement et à distance le cycle de vie des modules. Le système s'appuie sur une base de données regroupant des informations statiques sur l'infrastructure de l'environnement. Connaître tous les composants disponibles

en principe, même s'ils ne sont pas disponibles à un instant donné[1], permet à l'assistant de démarrer automatiquement un module requis pour rendre un service.

Les expérimentations que nous avons menées ont montré le bon fonctionnement de notre système. Pour évaluer le comportement appris, l'expérimentateur a évalué chaque association situation-action en indiquant si celle-ci est « la bonne ». Ces évaluations étaient en accord avec les récompenses données lors de l'apprentissage. Nous avons constaté que la note résultant de cette évaluation était croissante au fur et à mesure de l'apprentissage, jusqu'à atteindre une valeur stationnaire. En vérifiant le comportement appris à ce stade, nous avons constaté qu'il correspondait à ce que nous souhaitions faire apprendre au système. En faisant alors voir au système de nouvelles parties de l'environnement et de nouvelles récompenses, la croissance de la note a repris. Lorsque l'expert a soudainement changé sa stratégie de récompense, simulant un changement dans les préférences de l'utilisateur, la note a chuté, puis a progressivement ré-augmenté.

D'autre part, nous avons mené une enquête auprès d'utilisateurs potentiels de notre assistant. Par des questions et un dialogue ouvert, nous avons cherché à connaître l'opinion qu'a le public des systèmes d'informatique ambiante, l'usage que les personnes pourraient avoir d'un assistant tel que le nôtre, et leur avis sur son utilité. L'enquête nous a montré qu'il existe un profil de personnes trouvant une réelle utilité à notre assistant. Ce sont les personnes cognitivement surchargées, dont l'emploi du temps est très dense et dynamique. Ces personnes souhaitent être soulagées des tâches banales, elles accepteraient volontiers qu'un système automatique prenne en charge une partie des choses auxquelles elles doivent penser. D'autres personnes sont plutôt réticentes à un tel système. Elles craignent de devenir trop assistées et, petit à petit, devenir dépendantes de cet assistant. De manière générale, les utilisateurs ne souhaitent pas être trop sollicités ou dérangés par le système. L'hypothèse de l'apprentissage sur un entraînement simple est donc confirmée. D'autre part, les utilisateurs souhaitent également être engagés par le système. Certains ne veulent pas voir l'assistant s'effacer, mais désirent, au contraire, interagir avec celui-ci.

8.2 Perspectives

L'enquête que nous venons d'évoquer a révélé que tous les utilisateurs ne souhaitent pas avoir un assistant transparent, mais souhaitent pouvoir interagir avec lui. Il serait envisageable d'ajouter au système une phase optionnelle, éventuellement à l'initiative de l'utilisateur, de questions-réponses, effectuant un débriefing sur le comportement appris. Cette phase pourrait renforcer le modèle des récompenses et, par une étape d'apprentissage hors-ligne, d'intégrer ces spécifications au comportement. De plus, ceci permettrait de lever des doutes sur l'action à choisir dans des états dans lesquels le système ne sait pas comment agir.

Dans l'optique de cette phase de mise au point, et dans l'optique générale de pouvoir expliquer l'état courant du système à l'utilisateur, il serait intéressant d'étudier la meilleure manière de présenter cet état à l'usager. Les prédicats que nous utilisons sont humainement compréhensibles, mais l'état de l'assistant

[1] Car le composant recherché n'est ni installé, ni démarré sur un poste.

n'est pas très lisible pour autant. L'utilisateur peut le comprendre en y portant de l'attention, mais pas immédiatement, en y « jetant un coup d'œil ». Étudier une présentation synthétique de l'état courant, en sélectionnant les prédicats intéressants à afficher, rendrait le système d'autant plus compréhensible.

Une autre amélioration du système serait de le rendre plus dynamique et plus adaptable aux changements de l'environnement. Nous avons vu que l'apprentissage supervisé du modèle du monde est effectué régulièrement et à vie. Ainsi, si la dynamique des transitions de l'environnement change, notre système intégrera ce changement. Par contre, nous n'avons pas prévu un moyen d'adapter le système d'apprentissage aux changements dans les modules du système ambiant. Par exemple, si un nouveau capteur est ajouté en cours de route et si celui-ci détecte un type d'événements non encore pris en charge, il serait nécessaire d'ajouter un prédicat à la définition des états de notre agent d'apprentissage par renforcement. Afin de ne pas perdre toutes les connaissances déjà acquises et le comportement appris jusqu'alors, il serait nécessaire de définir une transformation traduisant toutes les connaissances que nous avions avec les anciens états en connaissances sur les nouveaux états. Lorsqu'il s'agit d'un remplacement de prédicat, il suffirait de trouver un parallèle entre arguments. Dans le cas d'un ajout de prédicat, il faut simplement ajouter ce prédicat à tous les états, avec des arguments vides. Enfin, dans le cas d'une suppression de prédicat, la procédure serait de fusionner les états qui n'avaient que ce prédicat de différent. Les nouvelles actions s'intégreraient par contre naturellement au système. La suppression d'une action nécessiterait une phase de nettoyage de la Q-table afin d'effacer toutes les entrées contenant l'action obsolète.

Par conséquent, s'adapter aux évolutions de l'ensemble des modules disponibles est faisable à condition de définir un nouveau prédicat dans le cas d'un ajout, ou de sélectionner le prédicat à supprimer ou encore de spécifier la modification d'un prédicat. Pour un expert, cette tâche est simple, mais nous ne souhaitons pas requérir l'intervention d'un expert au cours de la vie de l'assistant. Ainsi, il serait utile d'étudier des techniques guidant l'utilisateur dans la réalisation de cette tâche.

Enfin, une fonctionnalité additionnelle qui serait utile pour les usagers serait de créer des profils utilisateurs transportables d'un environnement à l'autre. Ainsi, si le bureau, la voiture et la maison d'une personne sont équipées de modules de notre système ambiant, il serait possible d'étendre le champ d'action de l'assistant à ces environnements. Le comportement devrait alors être représenté de manière compacte et transporté sur le périphérique mobile de l'utilisateur. L'état courant devrait alors inclure l'information sur l'environnement dans lequel se trouve l'utilisateur car certains services peuvent être différents selon si la personne se trouve chez elle ou bien au bureau. D'autres services, au contraire, seraient identiques. Ceci est faisable avec le mécanisme de généralisation et divisions d'états décrit sections 5.6.1.2 et 5.6.1.3. Pour réaliser ceci, nous envisageons l'ajout d'un prédicat représentant le lieu courant (bureau, maison ou voiture par exemple). Systématiquement généraliser la valeur de ce prédicat le rend inutile. Cette extension justifie pleinement les techniques de division d'états de la Q-table discutées dans ce manuscrit. Ce prédicat *lieu* est un exemple typique pour lequel il sera nécessaire de mettre en œuvre la division d'états ayant été au préalable fusionnés.

En conclusion, nous avons présenté une méthode qui pourrait rendre les environnements ubiquitaires intelligents. Leurs modèles de contexte seraient adap-

tatifs et évolutifs. Le prototype que nous avons mis en place montre la faisabilité de cette méthode. L'enquête menée auprès du grand public montre qu'il existe un intérêt pour ce genre de systèmes. L'étude de l'état de l'art nous montre que nous sommes véritablement entrain de nous diriger vers l'informatique ubiquitaire, voire même que nous y sommes déjà. Des applications telles que la nôtre ont bien un futur, à condition de respecter la vie privée des utilisateurs, de garantir la confidentialité des informations recueillies, et de ne pas détourner la détection des activités et des préférences pour une utilisation commerciale ou de surveillance.

Annexe A

Enquête grand public

A.1 Grille d'entretien

L'enquête grand public décrite dans le chapitre 3 suivait le guide ci-dessous, élaboré par Nadine Mandran, en collaboration des autres participants à l'enqête :

Introduction

Tout d'abord, je tiens à vous remercier d'avoir accepté de participer à cet entretien. Le sujet qui nous intéresse s'inscrit dans le cadre de la recherche en informatique. Votre participation va contribuer à l'enrichissement de deux thèses. Nous nous intéressons à ce que vous pensez des nouvelles technologies et ce qu'elles représentent pour vous. Surtout, laissez libre cours à votre imagination, votre avis nous intéresse d'autant plus que le terme de nouvelles technologies est une notion vague et mal définie.

Mesure de la connaissance sur les nouvelles technologies et leurs usages

- Aujourd'hui, on parle beaucoup de nouvelles technologies, qu'est-ce que cela évoque pour vous ? À quoi pensez-vous lorsque vous entendez le terme « nouvelles technologies » ?
- On dit aussi que les nouvelles technologies « sont partout ». Qu'en pensez-vous ? À votre avis sont-elles vraiment partout ?
- Avez-vous l'habitude d'utiliser des produits issus des nouvelles technologies ?
  - Si oui, poser les questions suivantes :
    - Lesquels ?
    - Pour quels types usages (fréquence et habitude) ?

Mesure de la perception et opinions sur les nouvelles technologies

- Quel est l'objet qui représente les plus les nouvelles technologies d'aujourd'hui ?
- Est-ce que vous l'utilisez ? Pourquoi ?

- Quels sont les avantages de cet outil ?
- Quels sont les inconvénients de cet outil ?
- Quels sont les autres objets qui, pour vous, représentent les nouvelles technologies ? *Pour l'enquêteur* : Noter la liste des outils.
- Quels sont les avantages de ces outils ?
- Quels sont les inconvénients de ces outils ?

Mesure de la connaissance sur les services en ligne et sur leur usage

- Avez-vous l'habitude d'utiliser un ordinateur ? Pour quels usages ?

Situation n°1 : si la personne a l'habitude d'utiliser un ordinateur à domicile ou au travail

- Est-ce que vous avez l'habitude d'utiliser Internet ?
- Est-ce que, par exemple, vous l'utilisez pour préparer des vacances, pour organiser un déplacement, etc. ?
- Quels sont les avantages de ces services en ligne ?
- Quels sont les défauts de ces services en ligne ?

Situation n°2 : la personne n'a pas l'habitude d'utiliser un ordinateur à domicile ou au travail

- Si j'ai bien noté ce que vous m'avez dit, vous utilisez rarement un ordinateur est-ce que malgré cela vous utilisez Internet ?
- Est-ce que, par exemple, vous l'utilisez pour préparer des vacances, pour organiser un déplacement, etc. ?
- Est-ce qu'une autre personne le fait pour vous ?
- Mais finalement, à votre avis, quels peuvent être les avantages de ces services en ligne ?
- Quels sont les défauts de ces services en ligne ?
- Et à votre avis, pourquoi Internet a-t-il autant de succès ?

Mesure de l'utilité des nouvelles technologies (en situation d'exception)

Situation n°1 : la personne connaît ces outils sous la forme « informatique ambiante »

- Parmi les objets que vous m'avez cités, lequel préférez-vous ? Pourquoi ?
- Lequel est pour vous le plus utile ? Pourquoi ?
- À quelle fréquence utilisez-vous ces outils ? (mesure du niveau d'utilisation)

Pour l'enquêteur : faire ici le résumé de ce qui a été dit afin de rompre le déroulement de l'entretien.

- Est-ce que vous pourriez me raconter une situation dans laquelle les nouvelles technologies vous ont été utiles ? Pourquoi ?
- Est-ce que vous en avez vécu d'autres ?
- Est-ce que vous pourriez me raconter une situation dans laquelle les nouvelles technologies ont été insuffisantes ?

– Quels outils auraient pu vous aider ?

Situation n°2 : la personne ne connaît pas vraiment ces outils

Pour l'enquêteur : montrer des photos de nouveaux outils : vitrines tactiles, GPS, etc., et donner la définition suivante : *Les nouvelles technologies sont des moyens récents pour communiquer, trouver et utiliser de l'information.*

– Est-ce que vous possédez des appareils de ce type chez vous ?
– Lesquels ?
– Niveau d'utilisation de ces outils ?
– Faire émerger les difficultés, lesquelles et pourquoi ?
– Faire raconter une histoire vécue, où une assistance technique aurait pu être utile ? Ou un des outils présentés aurait pu être utile ?

Le système COMPOSE

Pour l'enquêteur : à lire avec précision : Dans le cadre de nos travaux de recherche, nous développons un assistant personnel appelé Aladdin. Cet assistant a deux modes de fonctionnement, nous allons commencer par le premier mode. Aladdin peut connaître vos habitudes et vous proposer de manière automatique les outils que vous avez l'habitude d'utiliser. Il se charge des tâches répétitives que vous lui demandez. Par exemple, votre agenda le matin et la mise à jour de votre réveil le soir. Il sait aussi que le mercredi, en début d'après midi, vous téléphonez chez vous. Donc, le mercredi, de manière automatique en début d'après midi, il vous prévient de téléphoner chez vous et il compose même le numéro, à vous de décrocher. Il peut aussi prendre l'initiative et trouver de nouveaux services qui peuvent vous être utiles, comme par exemple vous transmettre un rappel de votre agenda si vous n'êtes pas devant votre ordinateur.

– Que pensez-vous de ce type d'assistant ?
– Pourquoi ?
– Est-ce que vous imagineriez d'autres situations où il serait intéressant ?
– Quels sont les avantages ?
– Quels sont les inconvénients ?

Maintenant, le deuxième mode de fonctionnement d'Aladdin. Il peut aussi, en cas de difficultés ou de problèmes, vous proposer un ensemble de services qui vous apporteront une solution pratique. Vous lui posez une question, et il vous fournit la solution la plus pertinente. Par exemple, votre fille vient de déchirer son costume de fée clochette pour le carnaval, vous devez en trouver un autre dans la soirée. L'assistant vous renvoie l'adresse et les horaires d'ouverture de ce type de magasins proches de votre domicile. Ou encore, il est 20h un soir d'hiver, demain vous partez en voiture en montagne. Vous voulez connaître l'état des routes. Votre assistant vous renvoie le service météo, le service bison futé et vous donne l'adresse de magasins d'équipements automobiles car la neige est annoncée.

– Que pensez-vous de ce type d'assistant ?
– Pourquoi ?
– Est-ce que vous imagineriez des situations dans lesquelles un tel système pourrait être intéressant ? Demander pour chacune d'elles comment il souhaite interagir avec COMPOSE (oral, clavier, souris, stylet, etc.).
– Situations à proposer :

  - à la maison ;
  - en vacances ;
  - en voiture pour des raisons personnelles ;
  - en voiture pour des raisons professionnelles ;
  - pour répondre à un problème administratif, fournitures de documents à l'école.
- Quels sont les avantages ?
- Quels sont les inconvénients ?

Acceptabilité pour COMPOSE

- Si l'assistant apprend mal, autrement dit, s'il n'arrive pas à vous proposer les bons outils au bon moment, quelle sera votre réaction ?
- Si l'assistant commet des erreurs, mais que vous savez qu'il est en train d'apprendre et de s'adapter à votre façon d'agir, quelle serait votre réaction et votre avis sur cet objet ? Lui laisseriez-vous une chance ?
- Seriez-vous prêt à passer un peu de temps pour répondre aux « questions » de l'assistant (questions sur ce qu'il a compris de vos habitudes), afin qu'il apprenne plus vite (que son comportement soit plus correct) ?
- Si l'assistant était capable d'expliquer ses décisions, qu'est-ce que cela vous apporterait-il ?
- Quels sont les avantages de ce type d'objet ?
- Quels sont les inconvénients ?
- Si la personne n'évoque pas le problème de la surveillance par le système, lui poser les questions suivantes :
- Dans le cas où le système apprend vos habitudes, est-ce que vous pensez que cela peut être fiable ?
- Feriez-vous confiance à ce système ?
- Par rapport à votre anonymat et liberté d'agir, comment trouvez-vous ce système ?

Fin

L'entretien est terminé, je vous remercie. Est-ce que vous avez des remarques à ajouter ? Ou des questions à me poser sur ce système ou sur nos recherches ?

Annexe B

Système ambiant

B.1 Schéma complet de la base de données

Dans la section 4.4, nous avons présenté la base de données utilisée par notre assistant ambiant. Voici, en figure B.1, le schéma complet de cette base de données.

B.2 Détail des modules du système ambiant

B.2.1 Capteurs

B.2.1.1 Modules Bluetooth

Dans cette section, nous allons donner plus de précision sur le module de détection de présence Bluetooth qui a été brièvement introduit section 4.5.1. Les pièces sont équipées de fiches Bluetooth USB et nous détectons la présence des appareils mobiles des utilisateurs (téléphones ou PDAs).

Étant donné que la valeur RSSI dépend du périphérique, il est préférable de calibrer chaque périphérique utilisé. Ce calibrage consiste à demander à l'utilisateur d'entrer dans un bureau avec son dispositif Bluetooth et à enregistrer les valeurs RSSI. La moyenne de ces mesures constitue le seuil particulier de cet appareil Bluetooth.

Les modules qui remplissent toutes ces fonctions sont les suivants :

BluetoothScanner ce service se contente de scruter la zone couverte par la fiche Bluetooth dans le but de détecter les périphériques présents et d'envoyer des événements d'apparition et disparition des périphériques.

BluetoothTracker ce service se base sur les événements reçus du `BluetoothScanner`. Si le périphérique capté est connu du système (s'il est enregistré dans la base de données), il est suivi par ce module. Le *tracker* s'y connecte et mesure le signal en permanence afin de déterminer si le périphérique est dans le bureau ou non. Le tracker envoie des événements d'entrée et de sortie des périphériques du bureau. La figure B.2 représente ce service, et sa connexion au `BluetoothScanner` via `remoteShell`.

BluetoothCalibrator ce service sert à calibrer un périphérique Bluetooth. Il mesure la puissance du signal pendant que l'utilisateur entre dans le

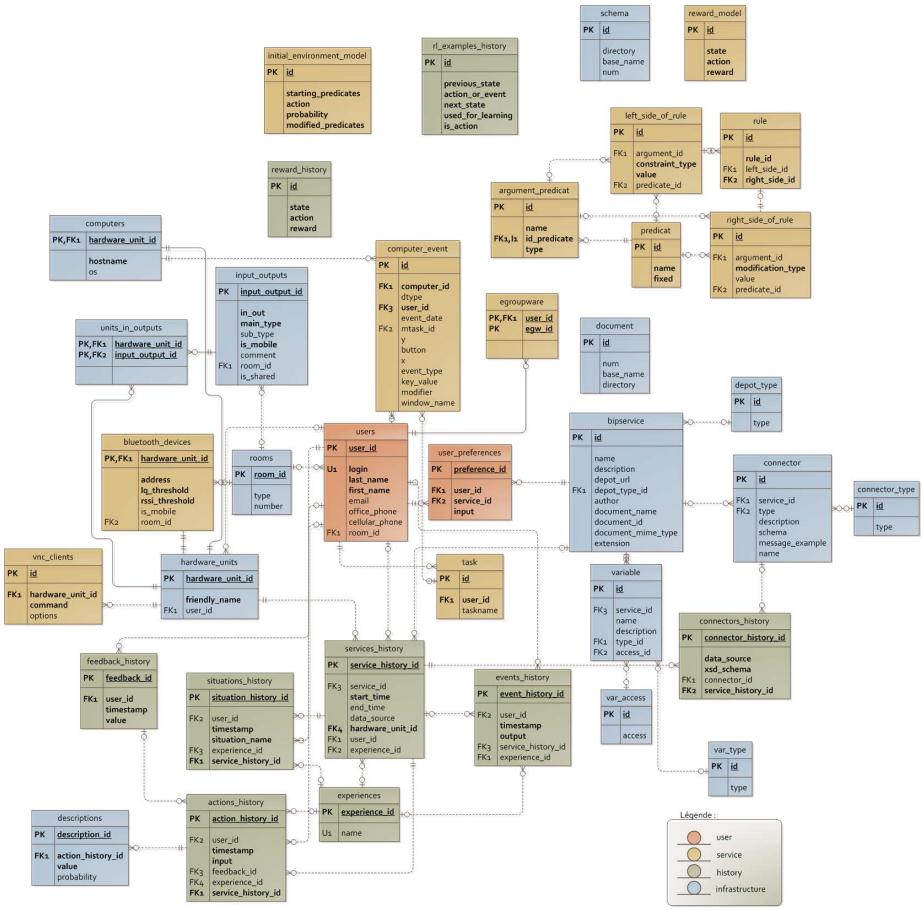

FIG. B.1 – La base de données complète utilisée par l'assistant personnel.

bureau afin d'enregistrer dans la base de données les valeurs de seuils de ce périphérique. Ceci n'a besoin d'être fait qu'une fois pour chaque périphérique. Ce service est montré sur la figure B.3, où l'on peut voir qu'il est connecté au service de synthèse vocale `Text2Speech` (dans le cas où ce dernier est disponible) afin de guider l'utilisateur dans la démarche de calibration.

Nous pouvons envisager de nous passer de cette étape de calibration. Les pistes à explorer sont la détermination d'une valeur par défaut avec laquelle le taux de fausse détections serait acceptable, ou bien une calibration « en laboratoire » pour différentes marques ou différents profils de puces Bluetooth. Nous pourrions encore effectuer un calibrage automatique en utilisant l'information fournie par un autre capteur de présence. Par exemple, si un tracker vidéo détecte l'entrée de l'utilisateur, le module de calibration Bluetooth pourrait effectuer le calibrage automatiquement à ce moment-là.

Les messages envoyés par le service `BluetoothTracker` (ainsi que par le service `BluetoothScanner`) sont de la forme :

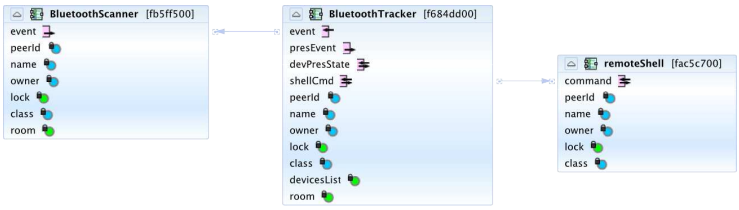

FIG. B.2 – Le service `BluetoothTracker` connecté au service `BluetoothScanner` et au service `remoteShell` qui a servi à démarrer le `BluetoothScanner`.

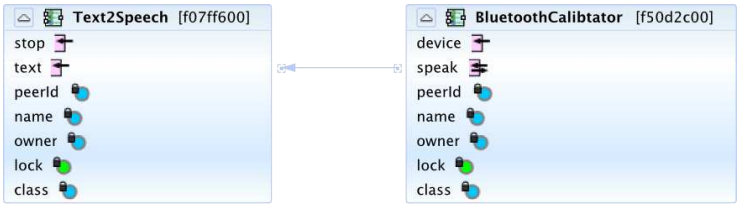

FIG. B.3 – Le service de calibration Bluetooth connecté au service de synthèse vocale.

```
<BTEvent valid="true">
   <address>00:12:47:C9:F2:AC</address>
   <type>entrance</type>
   <friendly-name>Sonia</friendly-name>
</BTEvent>
```

Le type de l'événement prend une valeur parmi cinq valeurs possibles. Les types `arrival` et `departure` ne peuvent être envoyés que par le service `BluetoothScanner` (connecteur `event`) et correspondent aux apparitions et disparitions (respectivement) de dispositifs dans la zone de couverture de la fiche Bluetooth. Les événements de types `entrance` et `exit` sont envoyés par le tracker lorsqu'un dispositif entre ou quitte le bureau (respectivement). Enfin, un événement du type `lost` peut être envoyé si la connexion avec le dispositif est perdue ou bien si l'utilisateur a souhaité interrompre le suivi en cliquant sur le bouton *Stop tracking* de l'interface figure 4.22.

B.2.1.2 UserLocalizationService

Ce service, montré figure B.4, permet de localiser l'utilisateur dans l'environnement ambiant. Il n'envoie pas d'événements mais répond à une requête de localisation (par le biais du connecteur `localize`). Il interroge en premier la base de données pour trouver le dernier événement d'entrée de l'utilisateur dans un bureau. Ensuite, il interroge le `BluetoothTracker` de ce bureau[1] afin de savoir si l'utilisateur est effectivement toujours dans ce bureau. S'il n'y est pas, alors le service répond que la localisation de l'utilisateur est inconnue. A priori, cela signifie qu'il n'est pas dans le bâtiment intelligent. L'utilisation de

[1] En listant la valeur de la variable `devicesList` de `BluetoothTracker` qui contient à chaque instant la liste des périphériques présents dans le bureau surveillé.

la base de données nous permet de gagner du temps et de ne pas interroger tous les trackers existants. Par ailleurs, ceci marchera de la même manière si nous disposions d'un autre détecteur de présence, comme par exemple un tracker vidéo ou un lecteur de tags RFiD. Ces différents trackers émettront tous les mêmes événements d'entrée et sortie et pourront également être interrogés sur la présence d'une cible dans leur zone de couverture.

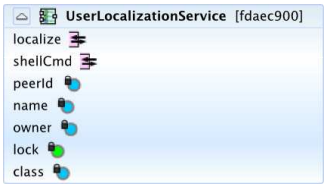

FIG. B.4 – Le service `UserLocalizationService` dans OMiSCIDGui.

B.2.1.3 Détection de nouveaux e-mails

Le service `MailService`, montré figure B.5, permet de surveiller l'arrivée de nouveaux messages électroniques dans la boite de courrier IMAP de l'utilisateur. Il envoie un événement sur son connecteur `listener` lors d'un nouveau message. Cet événement contient les paramètres du message (destinataire, expéditeur, sujet et corps). Le connecteur `newMessages` permet de recevoir les nouveaux courriers déjà présents dans la boîte de réception. Dès lors qu'un module se branche sur ce connecteur, il reçoit la liste de ces nouveaux e-mails.

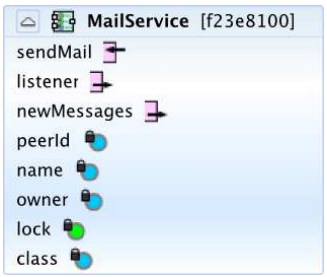

FIG. B.5 – Le service `MailService` dans OMiSCIDGui.

B.2.1.4 KdeEventsService

Ce service, montré figure B.6, détecte certains événements intéressants au niveau de KDE, donc de l'ordinateur de l'utilisateur. Toutes les minutes, ce service vérifie l'état des applications qui intéressent l'assistant, c'est-à-dire les rappels de l'agenda, l'état de la musique et de l'économiseur d'écran. Afin de simplifier l'implémentation, nous nous limitons aux applications accessibles via DCOP. DCOP, *(Desktop COmmunication Protocol)* est un système de communication léger entre les processus et les composants logiciels d'un système. Sa principale utilisation est de permettre aux différentes applications d'interagir et de partager des tâches complexes. DCOP est essentiellement un système de « contrôle à distance », qui peut faire profiter une application ou un script de l'aide des autres

applications. Il est construit au dessus du protocole d'échanges interclients de X.

`KdeEventsService` utilise le service `DcopService` pour exécuter des commandes dcop. De très nombreuses fonctionnalités des applications KDE sont accessibles via des commandes dcop. Ces commandes sont de la forme :

```
dcop [application] [objet] [fonction] [paramètres]
```

`KdeEventsService` connaît les commandes qui lui sont nécessaires, les envoie à `DcopService` (via son connecteur `queryDcop`) qui les exécute et renvoie la réponse. `KdeEventsService` analyse la réponse et en déduit éventuellement un événement à diffuser sur son connecteur `event`. Ces événements sont reçus par l'assistant qui les interprète comme changements de l'état de l'agent d'apprentissage par renforcement (voir la section 5.6.1).

Par exemple, pour connaître les rappels prévus dans l'agenda pour la journée, la commande est :

```
dcop korgac ac dumpAlarms
```

(la fonction appelée n'admettant pas de paramètres) et la réponse est de la forme :

```
AlarmDeamon::dumpAlarms() from Tue Apr 28 00:00:00 2009 to Tue
Apr 28 23:59:59 2009
  Seminaire d'équipe (Tue Apr 28 15:45:00 2009)
```

il suffit ensuite de vérifier si l'heure du rappel (15h45 dans l'exemple) est l'heure courante pour envoyer un événement avec les paramètres du rappel.

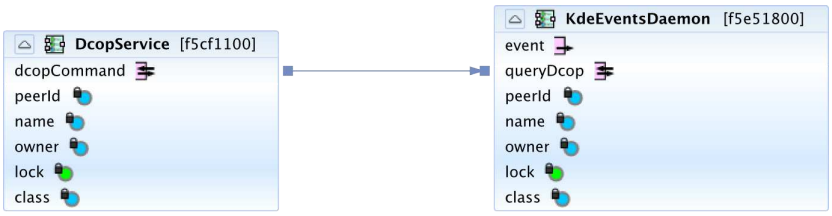

FIG. B.6 – Le service `KdeEventsService` connecté au service `DcopService` dans OMISCIDGui.

B.2.1.5 bipServeurTraceX

Ce service, montré figure B.7, écoute les événements du serveur X et envoie des événements par son connecteur `outputtraceX` correspondant à l'activité clavier, souris, et à l'application ayant le focus[2]. L'assistant utilise ces informations pour déterminer si l'utilisateur est actif sur son ordinateur, et également mettre à jour l'état de l'agent d'apprentissage par renforcement.

[2]Ce service est basé sur le logiciel *wmtrace* http://www.lri.fr/~chapuis/software/wmtrace/

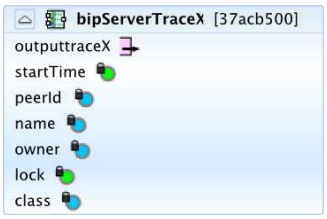

FIG. B.7 – Le service `bipServeurTraceX` dans OMiSCIDGui.

B.2.2 Effecteurs

B.2.2.1 Text2Speech

Ce module, montré figure B.8, intègre le synthétiseur vocal FreeTTS. afin de prononcer « à voix haute » un texte qui lui est transmis sur le connecteur d'entrée `text`. Le connecteur `stop` peut être utilisé afin d'interrompre la lecture en cours (ce qui peut être utile si le texte transmis est long).

FIG. B.8 – Le service `Text2Speech` dans OMiSCIDGui.

B.2.2.2 MessageService

Ce service, montré figure B.9, permet d'afficher un message écrit sur un écran d'une machine stationnaire ou mobile. Le message à afficher est reçu sur le connecteur `text`. Le format du message est le même que pour le module `Text2Speech` ci-dessus (section B.2.2.1), ce qui permet à l'assistant personnel de choisir au dernier moment la modalité pour transmettre le message. De plus, ceci traduit bien le fait que ces deux modules rendent un même service de différentes manières.

FIG. B.9 – Le service `MessageService` dans OMiSCIDGui.

B.2.2.3 MailService

Ce service, montré figure B.5, est à la fois un capteur et un effecteur car il permet non seulement de détecter l'arrivée d'un nouveau mail (connecteur `listener`), mais également d'en envoyer un (connecteur `sendMail`). Actuellement nous nous en servons pour transmettre un rappel à l'utilisateur si nous ne pouvons pas le faire par un autre moyen.

B.2.2.4 DcopService

Ce service, montré figure B.10, est utilisé par KdeEventsService (section B.2.1.4) pour détecter des événements informatiques (il joue alors le rôle d'un capteur), mais il peut également être utilisé pour lancer des applications et exécuter des commandes *dcop*, il joue alors le rôle d'un effecteur. En effet, ce service permet d'exécuter une commande *dcop* quelconque, ce qui lui donne un certain niveau de généricité. Nous pouvons, par exemple, lancer l'économiseur d'écran ou arrêter la musique. `DcopService` n'a qu'un seul connecteur entrée-sortie par lequel il reçoit une commande et envoie la réponse après exécution.

FIG. B.10 – Le service DcopService dans OMiSCIDGui.

Annexe C

Application de l'apprentissage par renforcement

C.1 Extrait de la Q-table

La figure C.1 montre un extrait de la Q-table. Pour plus de lisibilité, seuls les prédicats significatifs de l'état sont montrés.

C.2 Complexité – définitions

C.2.1 Classe PSPACE

La classe PSPACE est l'ensemble des problèmes de décision pouvant être résolus par une machine de Turing en utilisant une quantité polynomiale de mémoire et un temps illimité. La définition formelle est la suivante :

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$$

Un problème est PSPACE-*complet* s'il est parmi les problèmes les plus difficiles de la classe PSPACE. Plus formellement un problème A est dit PSPACE-*complet* si les deux conditions suivantes sont remplies.

1. Le problème A est dans la classe PSPACE, c'est-à-dire A $\in$ PSPACE,
2. tout problème PSPACE se réduit polynomialement à A, c'est-à-dire B $\leqslant_p$ A, $\forall$ B $\in$ PSPACE.

B $\leqslant_p$ A signifiant qu'il existe une réduction en temps polynomial et de type N : 1[1] de B vers A. Si seule la seconde condition est vérifiée, on dit que le problème A est PSPACE-*difficile*.

[1] C'est-à-dire une transformation d'instances d'un problème vers des instances d'un autre, transformation calculable en temps polynomial par une machine de Turing déterministe.

C.2.2 Classe P

Un problème de décision est dans P s'il peut être décidé sur une machine déterministe en temps polynomial par rapport à la taille de la donnée. On qualifie alors le problème de polynomial, c'est un problème de complexité $\mathcal{O}(n^k)$ pour un certain k.

Un problème de décision A est P-*complet* s'il appartient à la classe P et si tout problème dans P peut être réduit à A en utilisant une réduction appropriée.

C.2.3 Classe NC *(Nick's Class)*

La classe NC est la classe des problèmes qui peuvent être résolus en temps poly-logarithmique (c'est-à-dire résolus plus rapidement qu'il ne faut de temps pour lire séquentiellement leurs entrées) sur une machine parallèle ayant un nombre polynomial (c'est-à-dire raisonnable) de processeurs.

Un problème est dans NC s'il existe un algorithme pour le résoudre qui peut être parallélisé et qui gagne à l'être. C'est-à-dire, si la version parallèle de l'algorithme (s'exécutant sur plusieurs processeurs) est significativement plus efficace que la version séquentielle.

C.2.4 Classe NEXPTIME (ou NEXP)

La classe NEXPTIME est la classe des problèmes qui peuvent être résolus par une machine de Turing non-déterministe en un temps en $\mathcal{O}(2^{p(n)})$ pour un certain polynôme p et avec un espace mémoire infini. La définition formelle est la suivante :

$$\text{NEXPTIME} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k})$$

De même que pour les autres classes, un problème de décision A est NEXP-*complet* s'il appartient à la classe NEXP et s'il existe une réduction en temps polynomial et de type N : 1 de tout problème dans NEXP vers A. Un problème est dit NEXP-*difficile* s'il remplit seulement la dernière condition.

Remarque : la classe EXP est la classe des problèmes qui peuvent être résolus par une machine de Turing déterministe en un temps en $\mathcal{O}(2^{p(n)})$ pour un certain polynôme p.

C.2.5 Inclusions de classes

Il est prouvé que P $\subset$ PSPACE $\subset$ EXP $\subset$ NEXP et que NC $\subset$ P, mais on ne sait pas si P $\subset$ NC (et donc si NC = P). On conjecture que non, en supposant qu'il existe dans P des problèmes dont les solutions sont intrinsèquement non parallélisables.

Ligne	Extrait de l'état		Action	Q-valeur
1	entrance(friendlyName=<+>,isAlone=false,btAddress=<+>)	exit(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	pauseMusic&&unlockScreen	18,87308666
2	entrance(friendlyName=<+>,isAlone=false,btAddress=<+>)	exit(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	pauseMusic&&nothingAboutScreen	1,035952879
3	entrance(friendlyName=<+>,isAlone=false,btAddress=<+>)	exit(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	nothingAboutMusic&&nothingAboutScreen	0
4	entrance(friendlyName=<+>,isAlone=false,btAddress=<+>)	exit(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	nothing	-0,110758085
5	entrance(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	exit(friendlyName=<+>,isAlone=false,btAddress=<+>)	unpauseMusic&&nothingAboutScreen	1,499298106
6	entrance(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	exit(friendlyName=<+>,isAlone=false,btAddress=<+>)	unpauseMusic&&unlockScreen	0,423688391
7	entrance(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	exit(friendlyName=<+>,isAlone=false,btAddress=<+>)	nothingAboutMusic&&nothingAboutScreen	0
8	entrance(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	exit(friendlyName=<+>,isAlone=false,btAddress=<+>)	notForward&&notInform	0
9	entrance(friendlyName=<*>,isAlone=<*>,btAddress=<*>)	exit(friendlyName=<+>,isAlone=false,btAddress=<+>)	nothing	-0,157889655
10	inOffice(office=<+>,user=<+>)	computerState(isScreenLocked=true,machine=<+>,isMusicPaused=true)	unpauseMusic&&unlockScreen	8,22639668
11	inOffice(office=<+>,user=<+>)	computerState(isScreenLocked=true,machine=<+>,isMusicPaused=true)	unpauseMusic&&lockScreen	5,866907304
12	inOffice(office=<+>,user=<+>)	computerState(isScreenLocked=true,machine=<+>,isMusicPaused=true)	nothingAboutMusic&&unlockScreen	2,101099412
13	absent(user=<+>)	computerState(isScreenLocked=false,machine=<+>,isMusicPaused=false)	pauseMusic&&lockScreen	82,68477356
14	absent(user=<+>)	computerState(isScreenLocked=false,machine=<+>,isMusicPaused=false)	nothingAboutMusic&&unlockScreen	3,803470769
15	absent(user=<+>)	computerState(isScreenLocked=false,machine=<+>,isMusicPaused=false)	pauseMusic&&nothingAboutScreen	3,09128687

FIG. C.1 – Extrait de la Q-table.

Bibliographie

[Abowd et Mynatt, 2000] Gregory D. Abowd et Elizabeth D. Mynatt. « Charting past, present, and future research in ubiquitous computing ». *Dans ACM Trans. Comput.-Hum. Interact.*, tome 7, n° 1, ACM, 2000. URL http://portal.acm.org/citation.cfm?id=344949.344988#.

[Abowd *et al.*, 2002] Gregory D. Abowd, Elizabeth D. Mynatt et Tom Rodden. « The Human Experience ». *Dans IEEE Pervasive Computing*, tome 1, n° 1, pages 48–57, 2002.

[Adams *et al.*, 1993] Norman Adams, Rich Gold, Bill N. Schilit, Michael Tso et Roy Want. « An Infrared Network for Mobile Computers ». *Dans Proceedings USENIX Symposium on Mobile & Location-independent Computing*, pages 41–52. USENIX Association, Cambridge, Massachusetts, août 1993. URL http://www.ubiq.com/parctab/usmlic-93-adams-abstract.html.

[Aras *et al.*, 2007] Raghav Aras, Alain Dutech et François Charpillet. « Une méthode de programmation linéaire mixte pour les POMDP décentralisé à horizon fini ». *Dans $2^{èmes}$ Journées Francophones Planification, Décision, Apprentissage pour la conduite de systèmes – JFPDA 2007*. Grenoble / France, juillet 2007. URL http://hal.inria.fr/inria-00162469/en/.

[Assad *et al.*, 2007] Mark Assad, David Carmichael, Judy Kay et Bob Kummerfeld. « PersonisAD : Distributed, Active, Scrutable Model Framework for Context-Aware Services ». *Dans* Anthony LaMarca, Marc Langheinrich et Khai N. Truong, rédacteurs, *Proceedings of the 5th International Conference on Pervasive Computing, PERVASIVE 2007*, tome 4480 de *Lecture Notes in Computer Science*, pages 55 – 72. Springer, Toronto, Ontario, Canada, mai 2007. URL http://www.springerlink.com/content/kr08827t3h732343/?p=fab4d43a9a464124b05e3bbddf79964c&pi=3.

[Aström, 1965] K. J. Aström. « Optimal Control of Markov Decision Processes with Incomplete State Estimation ». *Dans Journal of Mathematical Analysis and Applications*, tome 10, pages 174–205, 1965. URL http://bibnetwiki.org/wiki/Optimal_Control_of_Markov_Decision_Processes_with_Incomplete_State_Estimation.

[Barton et Pierce, 2006] John Barton et Jeff Pierce. « Quantifying Magic In Ubicomp Systems Scenarios ». *Dans Position Paper for UbiSys 2006 Workshop at Ubicomp2006*. Orange County, California, septembre 2006. URL http://www.magic.ubc.ca/ubisys/positions/barton.pdf.

[Bell et Dourish, 2007] Genevieve Bell et Paul Dourish. « Yesterday's tomorrows : notes on ubiquitous computing's dominant vision ». *Dans Personal and Ubiquitous Computing*, tome 11, n° 2, pages 133–143, février 2007. URL http://www.springerlink.com/content/b22773n515483117.

[Bellotti et Edwards, 2001] Victoria Bellotti et Keith Edwards. « Intelligibility and accountability : human considerations in context-aware systems ». *Dans Human-Computer Interaction*, tome 16, n° 2, L. Erlbaum Associates Inc., décembre 2001. URL http://portal.acm.org/citation.cfm?id=1463113#.

[Bernstein *et al.*, 2009] Daniel S. Bernstein, Christopher Amato, Eric A. Hansen et Shlomo Zilberstein. « Policy Iteration for Decentralized Control of Markov Decision Processes ». *Dans Journal of AI Research (JAIR)*, tome 34, pages 89–132, février 2009. URL http://rbr.cs.umass.edu/~camato/decpomdp/publications.html.

[Bernstein *et al.*, 2002] Daniel S. Bernstein, Robert Givan, Neil Immerman et Shlomo Zilberstein. « The Complexity of Decentralized Control of Markov Decision Processes ». *Dans Mathematics of Operations Research*, tome 27, n° 4, pages 819–840, 2002. URL http://users.info.unicaen.fr/~gdibango/bibliography/search.php?lang=en&name=S.Zilberstein&article=TheComplexityofDecentralizedControlofMarkovDecisionProcesses.

[Bickmore et Mauer, 2006] Timothy Bickmore et Daniel Mauer. « Modalities for building relationships with handheld computer agents ». *Dans CHI '06 : CHI '06 extended abstracts on Human factors in computing systems*, pages 544–549. ACM, New York, NY, USA, 2006.

[Boutilier *et al.*, 1995] Craig Boutilier, Richard Dearden et Moises Goldszmidt. « Exploiting Structure in Policy Construction ». *Dans Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111. août 1995. URL http://citeseer.ist.psu.edu/248006.html.

[Brdiczka *et al.*, 2007] Oliver Brdiczka, James L. Crowley et Patrick Reignier. « Learning Situation Models for Providing Context-Aware Services ». *Dans Proceedings of HCI International*, tome 4555 de *LNCS*, pages 23–32. Springer Berlin / Heidelberg, août 2007. URL http://www.springerlink.com/content/571475u300245304/.

[Bridle et McCreath, 2004] Robert Bridle et Eric McCreath. « Improving the Learning Rate by Inducing a Transition Model ». *Dans AAMAS '04 : Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1330–1331. IEEE Computer Society, Washington, DC, USA, 2004. URL http://portal.acm.org/citation.cfm?id=1018934&coll=portal&dl=ACM&CFID=8769481&CFTOKEN=71481355#.

[Brown et Duguid, 1996] John Seely Brown et Paul Duguid. « Keeping it Simple ». *Dans Bringing Design to Software*, Terry Winograd, John Bennett, Laura De Young, and Bradley Hartfield, 1996. URL http://people.ischool.berkeley.edu/~duguid/SLOFI/Keeping_it_Simple.htm.

[Brown, 1996] P. J. Brown. « The Stick-e Document : a Framework for Creating Context-aware Applications ». *Dans Proceedings of EP'96, Palo Alto*, pages 259–272. also published in it EP–odd, janvier 1996. URL http://www.cs.kent.ac.uk/pubs/1996/396.

[Buffet, 2003] Olivier Buffet. « Une double approche modulaire de l'apprentissage par renforcement pour des agents intelligents adaptatifs ». Thèse de doctorat, Université Henri Poincaré, Nancy 1, septembre 2003. URL http://www.loria.fr/~buffet/?page=bib#Buffet-PhD03, laboratoire Lorrain de recherche en informatique et ses applications (LORIA).

[Byun et Cheverst, 2001] Hee Eon Byun et Keith Cheverst. « Exploiting user models and context-awareness to support personal daily activities ». *Dans Workshop in UM2001 on User Modeling for Context-Aware Applications*. 2001.

[Canny, 2006] John Canny. « The Future of Human-Computer Interaction ». *Dans ACM Queue*, tome 4, n° 6, pages 25–32, juillet – août 2006. URL http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=402.

[Chen *et al.*, 2004a] Harry Chen, Tim Finin et Anupam Joshi. « A Context Broker for Building Smart Meeting Rooms ». *Dans* Craig Schlenoff et Michael Uschold, rédacteurs, *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium*, pages 53–60. AAAI, AAAI Press, Menlo Park, CA, Stanford, California, mars 2004a. URL http://ebiquity.umbc.edu/paper/html/id/143/.

[Chen *et al.*, 2004b] Harry Chen, Tim Finin et Anupam Joshi. « An Ontology for Context-Aware Pervasive Computing Environments ». *Dans Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, tome 18, n° 3, pages 197–207, mai 2004b. URL http://ebiquity.umbc.edu/paper/html/id/60/.

[Cook *et al.*, 2003] D.J. Cook, M. Youngblood, III Heierman E.O., K. Gopalratnam, S. Rao, A. Litvin et F. Khawaja. « MavHome : an agent-based smart home ». *Dans Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 521–524. mars 2003. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1192783.

[Coutaz *et al.*, 2005] Joëlle Coutaz, James L. Crowley, Simon Dobson et David Garlan. « Context is key ». *Dans Commun. ACM*, tome 48, n° 3, pages 49–53, mars 2005. URL http://portal.acm.org/citation.cfm?id=1047671.1047703#.

[Coutaz et Rey, 2002] Joëlle Coutaz et Gaëtan Rey. « Foundations for a Theory of Contextors. » *Dans CADUI*, pages 13–34. 2002.

[Craik, 1943] Kenneth Craik. *The Nature of Exploration*. Cambridge University Press, Cambridge, England, 1943.

[Crowley, 2006] James L. Crowley. « Social Perception ». *Dans ACM Queue*, tome 4, n° 6, pages 35–43, juillet – août 2006. URL http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=405.

[Crowley *et al.*, 2002] James L. Crowley, Joëlle Coutaz, Gaëtan Rey et Patrick Reignier. « Perceptual Components for Context Aware Computing ». *Dans UBICOMP 2002, International Conference on Ubiquitous Computing*, pages 117–134. Springer Verlag, Goteborg, Sweden, septembre 2002. URL http://www-prima.imag.fr/prima/pub/Publications/2002/CCRR02/.

[Crowley *et al.*, 2009] James L. Crowley, Patrick Reignier et Rémi Barranquand. « Situation Models : A Tool for Observing and Understanding Activity ». *Dans Workshop on People Detection and Tracking, held at 2009 IEEE International Conference on Robotics and Automation, ICRA2009*. Kobe, Japan, mai 2009.

[Degris, 2007] Thomas Degris. « Apprentissage par Renforcement dans les Processus de Décision Markoviens Factorisés ». Thèse de doctorat, LIP6/AnimatLab, Université Pierre et Marie Curie, 2007. URL http://animatlab.lip6.fr/animatLab04-bib.html#degris07_these.

[Degris *et al.*, 2006a] Thomas Degris, Olivier Sigaud et Pierre-Henri Wuillemin. « Apprentissage de la structure des processus de décision markoviens factorisés pour l'apprentissage par renforcement ». *Dans Actes de la conférence JFPDA'06*, pages 89–96. Toulouse, 2006a. URL http://www.robot.jussieu.fr/?op=publication&lang=fr.

[Degris *et al.*, 2006b] Thomas Degris, Olivier Sigaud et Pierre-Henri Wuillemin. « Learning the structure of Factored Markov Decision Processes in reinforcement learning problems ». *Dans ICML '06 : Proceedings of the 23rd international conference on Machine learning*, pages 257–264. ACM Press, New York, NY, USA, 2006b.

[Dertouzos, 1999] M. L. Dertouzos. « The Future of Computing ». *Dans Scientific American*, tome 281, n° 2, Scientific American, New York, NY, ETATS-UNIS, 1999. URL http://www.sciam.com/article.cfm?id=the-future-of-computing.

[Dey et Abowd, 2000] Anind K. Dey et Gregory D. Abowd. « The Context Toolkit : Aiding the Development of Context-Aware Applications ». *Dans The Workshop on Software Engineering for Wearable and Pervasive Computing*. Limerick, Ireland, juin 2000. URL http://www.cs.cmu.edu/~anind/context.html.

[Dey *et al.*, 2001] Anind K. Dey, Gregory D. Abowd et Daniel Salber. « A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications ». *Dans Human-Computer Interaction*, tome 16, n° 2, Taylor & Francis, 2001. URL http://www.informaworld.com/10.1207/S15327051HCI16234_02.

[Diard, 2003] Julien Diard. « La carte bayésienne : un modèle probabiliste hiérarchique pour la navigation en robotique mobile ». Thèse de doctorat, Institut National Polytechnique de Grenoble, LEIBNIZ - GRAVIR / IMAG - INRIA Rhône-Alpes / GRAVIR-IMAG - E-MOTION, janvier 2003.

[Dibangoye *et al.*, 2009a] Jilles S. Dibangoye, Brahim Chaib-draa et Abdel-Illah Mouaddib. « Policy Iteration Algorithms for DEC-POMDPs with Discounted Rewards ». *Dans Proceedings of the Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM'09)*. 2009a. URL http://www.damas.ift.ulaval.ca/publication.php.

[Dibangoye *et al.*, 2009b] Jilles S. Dibangoye, Brahim Chaib-draa et Abdel-Illah Mouaddib. « Towards Policy Iteration Algorithms for Infinite Horizon DEC-POMDPs ». *Dans Proceedings of The 4th International AAMAS Workshop on MSDM*. mai 2009b. URL http://www.damas.ift.ulaval.ca/publication.php.

[Dibangoye *et al.*, 2009c] Jilles S. Dibangoye, Abdel-Illah Mouaddib et Brahim Chaib-draa. « Point-based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs ». *Dans Proceedings of The 8th International AAMAS Conference*. mai 2009c. URL http://www.damas.ift.ulaval.ca/publication.php.

[Dourish, 2004] Paul Dourish. « What we talk about when we talk about context ». *Dans Personal Ubiquitous Comput.*, tome 8, n° 1, pages 19–30, 2004. URL http://portal.acm.org/citation.cfm?id=970983.970985#.

[Ducatel *et al.*, 2001] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten et J-C. Burgelman. « Scenarios for Ambient Intelligence in 2010 ». Rapport technique, IST Advisory Group (ISTAG), IPTS-Seville, février 2001. URL http://cordis.europa.eu/ist/istag-reports.htm.

[Ducatel *et al.*, 2003] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten et J.-C. Burgelman. « Ambient Intelligence : From Vision to Reality ». Rapport technique, IST Advisory Group Draft Report, septembre 2003. URL http://cordis.europa.eu/ist/istag-reports.htm.

[Džeroski *et al.*, 1998] Sašo Džeroski, Luc De Raedt et Hendrik Blockeel. *Inductive Logic Programming*, tome 1446 de *Lecture Notes in Computer Science*, chapitre Relational reinforcement learning, pages 11–22. Springer Berlin / Heidelberg, 1998. URL http://www.springerlink.com/content/d4876533486t0h66.

[Džeroski *et al.*, 2001] Sašo Džeroski, Luc De Raedt et Kurt Driessens. « Relational Reinforcement Learning ». *Dans Machine Learning*, tome 43, n° 1–2, pages 7–52, avril 2001. URL http://www.springerlink.com/content/rn65l7p1p4733431.

[Edwards et Grinter, 2001] Keith W. Edwards et Rebecca E. Grinter. *Ubicomp 2001 : Ubiquitous Computing*, tome 2201 de *Lecture Notes in Computer Science*, chapitre At Home with Ubiquitous Computing : Seven Challenges, pages 256–272. janvier 2001. URL http://www.springerlink.com/content/h4nw9n0wftf3rp02/.

[Elrod *et al.*, 1992] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang et Brent Welch. « Liveboard : a large interactive display supporting group meetings, presentations, and remote collaboration ». *Dans CHI '92 : Proceedings of the SIGCHI conference on Human factors in computing systems*. 1992. URL http://portal.acm.org/citation.cfm?id=142750.143052#.

[Emonet *et al.*, 2006] Rémi Emonet, Dominique Vaufreydaz, Patrick Reignier et Julien Letessier. « O3MiSCID : an Object Oriented Opensource Middleware for Service Connection, Introspection and Discovery ». *Dans 1st IEEE International Workshop on Services Integration in Pervasive Environments*. juin 2006. URL http://www-prima.imag.fr/prima/pub/Publications/2006/EVRL06/.

[Gabillon *et al.*, 2008] Yoann Gabillon, Gaëlle Calvary et Humbert Fiorino. « Composing interactive systems by planning ». *Dans 4èmes journées Francophones Mobilité et Ubiquité (UbiMob'08)*. 2008. URL http://iihm.imag.fr/publication/GCF08a/.

[Geffner et Bonet, 1998] Héctor Geffner et Blai Bonet. « Solving Large POMDPs using Real Time Dynamic Programming ». *Dans Proc. AAAI Fall Symp. on POMDPs*. 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.6652.

[Gibson, 1999] William Gibson. « Interview on NPR's « Talk of the Nation » ». novembre 1999.

[Godoy et Amandi, 2005] Daniela Godoy et Analía Amandi. « User profiling for Web page filtering ». *Dans Internet Computing, IEEE*, tome 9, n° 4, pages 56–64, juillet – août 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1463162.

[Greenberg, 2001] Saul Greenberg. « Context as a dynamic construct ». *Dans Human-Computer Interaction*, tome 16, n° 2, L. Erlbaum Associates Inc., décembre 2001. URL http://portal.acm.org/citation.cfm?id=1463108.1463117#.

[Greenfield, 2006] Adam Greenfield. *Everyware : The Dawning Age of Ubiquitous Computing*. Peachpit Press, 2006. URL http://proquest.safaribooksonline.com/0321384016.

[Guestrin *et al.*, 2003] Carlos Guestrin, Daphne Koller, Chris Gearhart et Neal Kanodia. « Generalizing Plans to New Environments in Relational MDPs ». *Dans International Joint Conference on Artificial Intelligence (IJCAI-03)*. 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.7111.

[Gérard *et al.*, 2005] Pierre Gérard, Jean-Arcady Meyer et Olivier Sigaud. « Combining latent learning with dynamic programming in the modular anticipatory classifier system ». *Dans European Journal of Operational Research*, tome 160, n° 3, pages 614–637, 2005. URL http://www.sciencedirect.com/science/article/B6VCT-4B7N7X4-1/2/69e15fdace5401ed53a91df14571a4ca, decision Analysis and Artificial Intelligence.

[Hansen, 1998] Eric Anton Hansen. « Finite-memory control of partially observable systems ». Thèse de doctorat, University of Massachusetts Amherst, 1998. URL http://scholarworks.umass.edu/dissertations/AAI9909170/, director – Shlomo Zilberstein.

[Harper *et al.*, 2008] Richard Harper, Tom Rodden, Yvonne Rogers et Abigail Sellen. « Being Human : Human–Computer Interaction in the Year 2020 ». Rapport technique, Microsoft Research, 7 J J Thomson Avenue, Cambridge, CB3 0FB, England, avril 2008. URL http://research.microsoft.com/hci2020/.

[Hernandez *et al.*, 2004] Felix Hernandez, Elena Gaudioso et Jess G. Boticario. « A Reinforcement Learning Approach to Achieve Unobtrusive and Interactive Recommendation Systems for Web-Based Communities ». *Dans Adaptive Hypermedia and Adaptive Web-Based Systems*, tome 3137 de *Lecture Notes in Computer Science*, pages 409–412. Dpto. de Inteligencia Artificial, Universidad Nacional de Educacion a Distancia, C/ Juan del Rosal 16, 28040 Madrid, Spain, Springer Berlin / Heidelberg, Berlin / Heidelberg, décembre 2004. URL http://www.springerlink.com/content/4nyrq5t9el6ftj0v/.

[Hoey *et al.*, 2007] Jesse Hoey, Axel Von Bertoldi, Pascal Poupart et Alex Mihailidis. « Assisting persons with dementia during handwashing using a partially observable Markov decision process ». Rapport technique, National University of Singapore, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.64.6006.

[Isbell *et al.*, 2001] Charles Isbell, Christian R. Shelton, Michael Kearns, Satinder Singh et Peter Stone. « A social reinforcement learning agent ». *Dans AGENTS '01 : Proceedings of the fifth international conference on Autonomous agents*, pages 377–384. ACM Press, New York, NY, USA, 2001. URL http://portal.acm.org/citation.cfm?id=375735.376334#.

[Jaulmes *et al.*, 2005] Robin Jaulmes, Joelle Pineau et Doina Precup. « Learning in Non-Stationary Partially Observable Markov Decision Processes ». *Dans ECML Workshop on Reinforcement Learning in Non-Stationary Environments*. Porto, Portugal, 2005. URL http://www.cs.mcgill.ca/~jpineau/publications.html.

[José, 2008] Rui José. « Ubicomp 2.0 : From envisioning a future to being part of a new reality ». Plenary Talk at the UCAmI'08 conference, octobre 2008.

[Kaelbling, 2004] Leslie Pack Kaelbling. « Life-Sized Learning ». Lecture at CSE Colloquia, février 2004. URL http://www.uwtv.org/programs/displayevent.aspx?rID=3566.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman et Anthony R. Cassandra. « Planning and acting in partially observable stochastic domains ». *Dans Artificial Intelligence*, tome 101, n° 1–2, pages 99–134, 1998. URL http://portal.acm.org/citation.cfm?id=280145#.

[Kay, 1984] Alan Kay. « Computer Software ». *Dans Scientific American*, tome 251, n° 3, pages 52–59, septembre 1984. URL http://eric.ed.gov/ERICWebPortal/custom/portlets/recordDetails/detailmini.jsp?_nfpb=true&_&ERICExtSearch_SearchValue_0=EJ304669&ERICExtSearch_SearchType_0=no&accno=EJ304669.

[Kersting *et al.*, 2004] Kristian Kersting, Martijn Van Otterlo et Luc De Raedt. « Bellman goes relational ». *Dans ICML '04 : Proceedings of the twenty-first international conference on Machine learning*, tome 69 de *ACM International Conference Proceeding Series*, page 59. ACM, New York, NY, USA, 2004. URL http://portal.acm.org/citation.cfm?id=1015330.1015401#.

[Kidd *et al.*, 1999] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth Mynatt, Thad E. Starner et Wendy Newstetter. *Cooperative Buildings. Integrating Information, Organizations and Architecture*, tome 1670 de *Lecture Notes in Computer Science*, chapitre The Aware Home : A Living Laboratory for Ubiquitous Computing Research, pages 191–198. Springer Berlin / Heidelberg, 1999. URL http://www.springerlink.com/content/d598506140k50v26/.

[Kozierok et Maes, 1993] Robyn Kozierok et Pattie Maes. « A learning interface agent for scheduling meetings ». *Dans IUI '93 : Proceedings of the 1st international conference on Intelligent user interfaces*, pages 81–

88. ACM Press, New York, NY, USA, 1993. URL http://portal.acm.org/citation.cfm?id=169908&coll=portal&dl=ACM#.

[Kuvayev et Sutton, 1996] Leonid KUVAYEV et Richard SUTTON. « Model-Based Reinforcement Learning with an Approximate, Learned Model Leonid Kuvayev Rich Sutton ». *Dans in Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, pages 101–105. 1996. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.505.

[Leahu *et al.*, 2008] Lucian LEAHU, Phoebe SENGERS et Michael MATEAS. « Interactionist AI and the promise of ubicomp, or, how to put your box in the world without putting the world in your box ». *Dans UbiComp '08 : Proceedings of the 10th international conference on Ubiquitous computing*, tome 344, pages 134–143. ACM, New York, NY, USA, 2008. URL http://portal.acm.org/citation.cfm?id=1409654&dl=acm&coll=#.

[Lebeltel, 1999] Olivier LEBELTEL. « Programmation Bayésienne des Robots ». Thèse de doctorat, Institut National Polytechnique de Grenoble, Laboratoire Leibniz, octobre 1999. URL http://tel.archives-ouvertes.fr/tel-00011636/en/.

[Letessier et Vaufreydaz, 2005] Julien LETESSIER et Dominique VAUFREYDAZ. « Draft spec : BIP/1.0 – A Basic Interconnection Protocol for Event Flow Services ». http ://www-prima.imag.fr/prima/pub/Publications/2005/LV05/, 2005. URL http://www-prima.imag.fr/prima/pub/Publications/2005/LV05/.

[Lin, 1990] Long-Ji LIN. « Self-improving reactive agents : case studies of reinforcement learning frameworks ». *Dans Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 297–305. MIT Press, Cambridge, MA, USA, 1990. URL http://portal.acm.org/citation.cfm?id=116551.

[Littman, 1994] Michael L. LITTMAN. « Markov Games as a Framework for Multi-Agent Reinforcement Learning ». *Dans Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994. URL http://citeseer.ist.psu.edu/littman94markov.html.

[Lovejoy, 1991] William S. LOVEJOY. « Computationally Feasible Bounds for Partially Observed Markov Decision Processes ». *Dans Operations Research*, tome 39, n° 1, pages 162–175, janvier – février 1991. URL http://www.jstor.org/pss/171496.

[Maes, 1994] Pattie MAES. « Agents that reduce work and information overload ». *Dans Commun. ACM*, tome 37, n° 7, pages 30–40, juillet 1994. URL http://portal.acm.org/citation.cfm?id=176792#.

[Maes et Kozierok, 1993] Pattie MAES et Robyn KOZIEROK. « Learning Interface Agents ». *Dans Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 459–464. The AAAI Press, juillet 1993. URL http://www.aaai.org/Library/AAAI/1993/aaai93-069.php.

[Matignon *et al.*, 2007] Laëtitia MATIGNON, Guillaume LAURENT et Nadine Le FORT-PIAT. « Un algorithme décentralisé d'apprentissage par renforcement multi-agents coopératifs : le Q-Learning Hystérétique ». *Dans* Cépaduès EDITIONS, rédacteur, *2èmes Journées Francophones Planification, Décision, Apprentissage pour la conduite de Systèmes. JFPDA'07*, pages

115–121. Grenoble France, juillet 2007. URL http://hal.archives-ouvertes.fr/hal-00161653/en.

[Monahan, 1982] George E. Monahan. « A Survey of Partially Observable Markov Decision Processes : Theory, Models, and Algorithms ». *Dans Management Science*, tome 28, n° 1, pages 1–16, janvier 1982. URL http://mansci.journal.informs.org/cgi/content/abstract/28/1/1.

[Moore et Atkeson, 1993] Andrew W. Moore et Christopher G. Atkeson. « Prioritized sweeping : Reinforcement learning with less data and less time ». *Dans Machine Learning*, tome 13, pages 103–130, 1993. URL http://www.springerlink.com/content/r0t830035u571jt7/.

[Morihiro *et al.*, 2004] Koichiro Morihiro, Nobuyuki Matsui et Haruhiko Nishimura. « Reinforcement learning using chaotic exploration in maze world ». *Dans SICE 2004 Annual Conference*, tome 2, pages 1368–1371, août 2004. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1491636.

[Negroponte, 1972] Nicholas Negroponte. *The architecture machine : toward a more human environment*. MIT ; 216. MIT Press, 1972. URL http://catalogue.nla.gov.au/Record/2684154.

[Nonogaki et Ueda, 1991] Hajime Nonogaki et Hirotada Ueda. « FRIEND21 project : a construction of 21st century human interface ». *Dans CHI '91 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 407–414. ACM, New York, NY, USA, 1991. URL http://portal.acm.org/citation.cfm?id=108978&dl=ACM&coll=portal#.

[Norman, 1999] Donald A. Norman. *The Invisible Computer*. MIT Press, 1999. URL http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=5160.

[Nurmi et Floréen, 2004] Petteri Nurmi et Patrik Floréen. « Reasoning in Context-Aware Systems ». Helsinki Institute for Information Technology (HIIT) Basic Research Unit (BRU), 2004. URL http://www.cs.helsinki.fi/u/ptnurmi/papers/positionpaper.pdf.

[Paduraru, 2007] Cosmin Paduraru. « Planning with Approximate and Learned Models of Markov Decision Processes ». Thèse de maître, University of Alberta, 2007. URL http://www.cs.mcgill.ca/~cpadur/Site/Publications.html.

[Papadimitriou et Tsitsiklis, 1987] Christos H. Papadimitriou et John N. Tsitsiklis. « The Complexity of Markov Decision Processes ». *Dans Mathematics of Operations Research*, tome 12, n° 3, pages 441–450, août 1987. URL http://www.mit.edu/~jnt/publ.html.

[Pascoe *et al.*, 2007] Jason Pascoe, Kirsten Thomson et Helena Rodrigues. *On the Move to Meaningful Internet Systems 2007 : OTM 2007 Workshops*, chapitre Context-Awareness in the Wild : An Investigation into the Existing Uses of Context in Everyday Life, pages 193–202. Lecture Notes in Computer Science. novembre 2007. URL http://www.springerlink.com/content/u4u770m8n31l7050.

[Peng et Williams, 1993] Jing Peng et Ronald J. Williams. « Efficient Learning and Planning Within the Dyna Framework ». *Dans Adaptive Behavior*, pages 437–454. 1993. URL http://adb.sagepub.com/cgi/content/abstract/1/4/437.

[Pineau *et al.*, 2003] Joelle Pineau, Geoffrey Gordon et Sebastian Thrun. « Point-based value iteration : An anytime algorithm for POMDPs ». *Dans International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025–1032. août 2003. URL http://www-old.ri.cmu.edu/pubs/pub_4826.html#text_ref.

[Pollack *et al.*, 2003] Martha E. Pollack, Laura Brown, Dirk Colbry, Colleen E. McCarthy, Cheryl Orosz, Bart Peintner, Sailesh Ramakrishnan et Ioannis Tsamardinos. « Autominder : an intelligent cognitive orthotic system for people with memory impairment ». *Dans Robotics and Autonomous Systems*, tome 44, n° 3–4, pages 273–282, septembre 2003. URL http://www.sciencedirect.com/science/article/B6V16-48Y6T09-2/2/2cdf7370d62b92a5eeb4c6ef0e67cb44, best papers presented at IAS-7.

[Ranganathan *et al.*, 2004] Anand Ranganathan, Jalal Al-Muhtadi et Roy H. Campbell. « Reasoning about Uncertain Contexts in Pervasive Computing Environments ». *Dans IEEE Pervasive Computing*, tome 3, n° 2, pages 62–70, avril 2004. URL http://portal.acm.org/citation.cfm?id=998570&dl=&coll=#.

[Reignier *et al.*, 2006] Patrick Reignier, Sofia Zaidenberg, Rémi Emonet, Dominique Vaufreydaz et Julien Letesssier. « jOMiSCID, un intergiciel sous OSGi pour l'informatique ubiquitaire ». *Dans Atelier OSGi, 3ème Journées Francophones Mobilité et Ubiquité*. Paris, France, septembre 2006.

[Remagnino et Foresti, 2005] P. Remagnino et G.L. Foresti. « Ambient Intelligence : A New Multidisciplinary Paradigm ». *Dans Systems, Man and Cybernetics, Part A, IEEE Transactions on*, tome 35, n° 1, pages 1–6, janvier 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1369340&isnumber=29969.

[Rey et Coutaz, 2004] Gaëtan Rey et Joëlle Coutaz. « The Contextor Infrastructure for Context-Aware Computing ». *Dans Component-oriented Approaches to Context-aware Computing held in conjunction with ECOOP'04*. Oslo, juin 2004.

[Rey *et al.*, 2002] Gaëtan Rey, Joëlle Coutaz et James L. Crowley. « The Contextor : a computational Model for Contextural Information ». *Dans Wokshop Building Bridges Interdisciplinary Context-Sensitive Computing*. 2002. URL http://iihm.imag.fr/rey/publications.html.

[Richard et Yamada, 2007] Nadine Richard et Seiji Yamada. *Advances in Ambient Intelligence*, tome 164 de *Frontiers in Artificial Intelligence and Applications*, chapitre Two Issues for an Ambient Reminding System : Context-Awareness and User Feedback, pages 123–142. IOS Press, Nieuwe Hemweg 6B, 1013 BG Amsterdam, The Netherlands, novembre 2007. URL http://www.iospress.nl/loadtop/load.php?isbn=9781586038007.

[Ricquebourg *et al.*, 2006a] Vincent Ricquebourg, David Menga, Laurent Delahoche, Bruno Marhic, David Durand et Christophe Logé. « Architecture de perception de contexte orientée service pour l'habitat communicant ». *Dans 3èmes Journées Francophones Mobilité et Ubiquité (Ubimob'06)*. ACM, septembre 2006a. URL http://afihm.enst.fr/ubimob06/.

[Ricquebourg *et al.*, 2006b] Vincent RICQUEBOURG, David MENGA, David DURAND, Bruno MARHIC, Laurent DELAHOCHE et Christophe LOGÉ. « The Smart Home Concept : our immediate future ». *Dans* IEEE Industrial Electronics SOCIETY, rédacteur, *Proceedings of the First International Conference on E-Learning in Industrial Electronics*. Hammamet - Tunisia, décembre 2006b. ICELIE'06.

[Rogers, 2006] Yvonne ROGERS. « Moving on from Weiser's Vision of Calm Computing : Engaging UbiComp Experiences ». *Dans UbiComp 2006*, pages 404–421. Springer, 2006.

[Roman *et al.*, 2002] Manuel ROMAN, Christopher K. HESS, Renato CERQUEIRA, Anand RANGANATHAN, Roy H. CAMPBELL et Klara NAHRSTEDT. « Gaia : A Middleware Infrastructure to Enable Active Spaces ». *Dans IEEE Pervasive Computing*, pages 74–83, octobre – décembre 2002. URL http://choices.cs.uiuc.edu/~ranganat/publications.htm.

[Roy et Gordon, 2003] Nicholas ROY et Geoffrey GORDON. « Exponential family PCA for belief compression in POMDPs ». *Dans NIPS*, pages 1043–1049. MIT Press, 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.7155.

[Ryan *et al.*, 1998] N. S. RYAN, J. PASCOE et D. R. MORSE. « Enhanced Reality Fieldwork : the Context-aware Archaeological Assistant ». *Dans* V. GAFFNEY, M. VAN LEUSEN et S. EXXON, rédacteurs, *Computer Applications in Archaeology 1997*, British Archaeological Reports. Tempus Reparatum, Oxford, octobre 1998. URL http://www.cs.kent.ac.uk/pubs/1998/616.

[Schiaffino et Amandi, 2004] Silvia SCHIAFFINO et Analía AMANDI. « User–interface agent interaction : personalization issues ». *Dans International Journal of Human-Computer Studies*, tome 60, n° 1, pages 129–148, janvier 2004. URL http://www.sciencedirect.com/science/article/B6WGR-4B41VXD-3/2/ed547c9fb53a34e4b52043c449a97f17.

[Schiaffino et Amandi, 2006] Silvia SCHIAFFINO et Analía AMANDI. « Polite personal agent ». *Dans Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]*, tome 21, n° 1, pages 12 – 19, janvier – février 2006. URL http://ieeexplore.ieee.org/search/srchabstract.jsp?arnumber=1588797&isnumber=33480&punumber=9670&k2dockey=1588797@ieeejrns&query=%28+personal+agent%3Cin3Emetadata+%29&pos=0.

[Schilit *et al.*, 1994] Bill SCHILIT, Norman ADAMS et Roy WANT. « Context-aware computing applications ». *Dans In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

[Schilit et Theimer, 1994] Bill N. SCHILIT et Marvin M. THEIMER. « Disseminating active map information to mobile hosts ». *Dans Network, IEEE*, tome 8, n° 5, pages 22–32, septembre – octobre 1994. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=313011.

[Seuken et Zilberstein, 2008] Sven SEUKEN et Shlomo ZILBERSTEIN. « Formal models and algorithms for decentralized decision making under uncertainty ». *Dans Autonomous Agents and Multi-Agent Systems*, tome 17, n° 2, pages 190–250, février 2008. URL http://www.springerlink.com/content/w2l58437828m1055/.

[Shani *et al.*, 2007] Guy Shani, Ronen I. Brafman et Solomon E. Shimony. « Forward search value iteration for POMDPs ». *Dans IJCAI*. IJCAI, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.74.4567.

[Shankar et Louis, 2005] Anil Shankar et Sushil Louis. « Learning classifier systems for user context learning ». *Dans Evolutionary Computation, 2005. The 2005 IEEE Congress on*, tome 3, pages 2069–2075. septembre 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1554950.

[Siegemund et Flörkemeier, 2003] Frank Siegemund et Christian Flörkemeier. « The Smart Medicine Cabinet ». *Dans Systems Group, Institute for Pervasive Computing, Swiss Federal Institute of Technology (ETH)*. 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.3873.

[Sigaud, 2007] Olivier Sigaud. « Les systèmes de classeurs : Un état de l'art ». *Dans Revue d'intelligence artificielle*, tome 21, n° 1, Lavoisier, Paris, FRANCE (1987) (Revue), 2007. URL http://cat.inist.fr/?aModele=afficheN&cpsidt=18583982.

[Sigaud et Gérard, 2005] Olivier Sigaud et Pierre Gérard. « L'apprentissage par renforcement indirect dans les systèmes de classeurs ». *Dans Journal électronique d'intelligence artificielle*, tome 4, n° 19, 2005. URL http://jedai.afia-france.org/detail.php?PaperID=19.

[Silver *et al.*, 2007] David Silver, Richard Sutton et Martin Müller. « Reinforcement learning of local shape in the game of Go ». *Dans IJCAI 2007*. 2007. URL http://www.citeulike.org/group/5884/article/2990581.

[Singh, 1992] Satinder P. Singh. « Reinforcement Learning with a Hierarchy of Abstract Models ». *Dans Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 202–207. 1992. URL http://eprints.kfupm.edu.sa/61885/.

[Singh *et al.*, 1994] Satinder P. Singh, Tommi Jaakkola et Michael I. Jordan. « Learning Without State-Estimation in Partially Observable Markovian Decision Processes ». *Dans Proceedings of the Eleventh International Conference on Machine Learning*, pages 284–292. Morgan Kaufmann, 1994. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.9833.

[Smallwood et Sondik, 1973] Richard D. Smallwood et Edward J. Sondik. « The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon ». *Dans Operations Research*, tome 21, n° 5, pages 1071–1088, septembre – octobre 1973. URL http://www.jstor.org/pss/168926.

[Smart et Kaelbling, 2002] William D. Smart et Leslie Pack Kaelbling. « Effective Reinforcement Learning for Mobile Robots ». *Dans Proceedings of IEEE International Conference on Robotics and Automation (ICRA 2002)*, tome 4, pages 3404–3410. mai 2002. URL http://www.cse.wustl.edu/~wds/papers.php?detail#icra2002.

[Smith et Simmons, 2005] Trey Smith et Reid G. Simmons. « Point-Based POMDP Algorithms : Improved Analysis and Implementation ». *Dans Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*. 2005. URL http://www.cs.cmu.edu/~trey/pubs/b2hd-smith05hsvi.html.

[Spaan et Vlassis, 2005] Matthijs T. J. Spaan et Nikos Vlassis. « Perseus : Randomized point-based value iteration for POMDPs ». *Dans Journal of Artificial Intelligence Research*, tome 24, pages 195–220, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.9217.

[Stanfill et Waltz, 1986] Craig Stanfill et David Waltz. « Toward memory-based reasoning ». *Dans Commun. ACM*, tome 29, n° 12, pages 1213–1228, décembre 1986. URL http://portal.acm.org/citation.cfm?doid=7902.7906#.

[Suchman, 1987] Lucy A. Suchman. *Plans and situated actions : the problem of human-machine communication*. Cambridge University Press, New York, NY, USA, 1987. URL http://portal.acm.org/citation.cfm?id=38407#.

[Sutton, 1990] Richard S. Sutton. « Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming ». *Dans Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. URL http://portal.acm.org/citation.cfm?id=102055.

[Sutton, 1991] Richard S. Sutton. « Dyna, an integrated architecture for learning, planning, and reacting ». *Dans SIGART Bull.*, tome 2, n° 4, pages 160–163, 1991. URL http://portal.acm.org/citation.cfm?id=122377&jmp=cit&coll=GUIDE&dl=ACM&CFID=15151515&CFTOKEN=6184618.

[Sutton et Barto, 1998] Richard S. Sutton et Andrew G. Barto. *Reinforcement Learning : An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, mars 1998.

[Szer et Charpillet, 2006] Daniel Szer et François Charpillet. « Programmation dynamique à base de points pour la résolution des DEC-POMDPs ». *Dans 14èmes Journées Francophones sur les Systèmes Multi-Agents – JFSMA'2006*. Annecy, France, 2006. URL http://hal.inria.fr/inria-00104450/en/.

[Taylor *et al.*, 2007] Alex S. Taylor, Richard Harper, Laurel Swan, Shahram Izadi, Abigail Sellen et Mark Perry. « Homes that make us smart ». *Dans Personal and Ubiquitous Computing*, tome 11, n° 5, Springer London, juin 2007. URL http://www.springerlink.com/content/ar217785q7555282.

[Thomaz *et al.*, 2006] Andrea L. Thomaz, Guy Hoffman et Cynthia Breazeal. « Reinforcement Learning with Human Teachers : Understanding How People Want to Teach Robots ». *Dans Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 352–357. septembre 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4107833.

[Vallée *et al.*, 2005] M. Vallée, F. Ramparany et L. Vercouter. « Dynamic Service Composition in Ambient Intelligence Environments : a Multi-Agent Approach ». *Dans Proceeding of the First European Young Researcher Workshop on Service-Oriented Computing*. Leicester, UK, avril 2005. URL http://www.hitech-projects.com/euprojects/amigo/publicaties.htm.

[Walker, 2000] Marilyn A. Walker. « An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email ».

Dans Journal of Artificial Intelligence Research, tome 12, pages 387–416, 2000.

[Walker *et al.*, 2008] Trevor Walker, Lisa Torrey, Jude Shavlik et Richard Maclin. *Inductive Logic Programming*, tome 4894 de *Lecture Notes in Computer Science*, chapitre Building Relational World Models for Reinforcement Learning, pages 280–291. Springer Berlin / Heidelberg, février 2008. URL http://www.springerlink.com/content/45w66n614h711356.

[Watkins, 1989] CJCH Watkins. « Learning from Delayed Rewards ». Thèse de doctorat, University of Cambridge, England, mai 1989. URL http://www.citeulike.org/user/martinh/article/549749.

[Weber *et al.*, 2005] Werner Weber, Jan M. Rabaey et Emile Aarts. *Ambient Intelligence*, chapitre Ambient Intelligence Research in HomeLab : Engineering the User Experience, pages 49–61. Springer Berlin Heidelberg, décembre 2005. URL http://www.springerlink.com/content/rv224p604714w256/.

[Weiser, 1991] Mark Weiser. « The computer for the 21st century ». *Dans Scientific American*, tome 265, n° 3, pages 66–75, septembre 1991.

[Weiser, 1993] Mark Weiser. « Some computer science issues in ubiquitous computing ». *Dans Commun. ACM*, tome 36, n° 7, pages 75–84, juillet 1993. URL http://portal.acm.org/citation.cfm?id=159617#.

[Weiser, 1994] Mark Weiser. « The world is not a desktop ». *Dans Interactions*, tome 1, n° 1, pages 7–8, janvier 1994. URL http://portal.acm.org/citation.cfm?id=174801#.

[Yan et Selker, 2000] Hao Yan et Ted Selker. « Context-aware office assistant ». *Dans IUI '00 : Proceedings of the 5th international conference on Intelligent user interfaces*, pages 276–279. ACM Press New York, NY, USA, New Orleans, Louisiana, United States, janvier 2000. URL http://portal.acm.org/citation.cfm?id=325872\&dl=ACM\&coll=ACM.