Document :	*Expérimentation et évaluation du scénario prospectif*
Sous-tâches :	6.2
Numéro des Livrables :	D6.2
Date	25/05/2012
Rédacteurs :	*Joëlle Coutaz (LIG), Alexandre Demeure (LIG), Emeric Fontaine (LIG), Nadine Mandran (LIG), Teresa Colombi (LudoTIC), Aurore Russo (LudoTIC), Leonid Synyukov (LudoTIC), Stéphane Lavirotte (I3S)*
Coordinateurs :	Joëlle Coutaz

Résumé

Ce livrable (D6.2) présente les résultats de l'évaluation de l'acceptabilité de KISS (Knit your Ideas into Smart Spaces). *KISS (Knit Your Ideas into Smart Spaces) est un outil de programmation et de mise au point* dont le langage de programmation est de type déclaratif orienté règles, avec potentiel d'égale opportunité syntaxique entre langue française pseudonaturelle (LPN) et langage visuel iconique. La technique d'interaction de construction des programmes LPN s'appuie sur l'utilisation de menus dont les options sont calculées dynamiquement assurant ainsi la découverte progressive du langage ainsi que l'extensibilité et la correction syntaxique et sémantique des programmes. La mise au point peut se pratiquer, au choix, dans le monde physique ou dans un monde dual numérique.

Sur le plan technique, KISS est l'un des services de l'infrastructure d'exécution CONTINUUM qui, à son tour, s'appuie sur l'intergiciel WCOMP pour la composition et la reconfiguration dynamiques. Composition et reconfiguration sont exprimées en Aspect d'Assemblage (AA). KISS, qui joue le rôle de méta-IHM tactique dans l'infrastructure CONTINUUM, traduit les programmes de l'utilisateur en plans d'adaptation sous la forme de couples « situation-AA ». Kiss fait partie des rares exemples de l'état de l'art où les phases de conception et d'exécution peuvent être perçues comme confondues.

KISS a été évalué par 13 sujets représentatifs dans DOMUS, l'appartement intelligent du Laboratoire d'Informatique de Grenoble (LIG). Ces sujets ont programmé avec succès une séquence clef réaliste du scénario prospectif de CONTINUUM. Il en ressort que le contrôle de l'habitat avec KISS est une expérience appréciée, que le langage pseudonaturel et l'habitat virtuel sont accessibles et que leurs rôles respectifs sont bien compris. Les difficultés rencontrées confirment nos hypothèses sur la nécessité de pousser plus avant le multisyntaxe assorti de techniques d'interaction multimodales.

D6.2 rappelle les motivations qui nous ont conduits à développer KISS (essentiellement, redonner le pouvoir à l'utilisateur final), puis consigne quelques définitions en lien avec KISS, de même le positionnement de KISS dans l'architecture Continuum. KISS est ensuite décrit du point de vue de l'utilisateur final. Les adaptations techniques nécessaires à l'expérimentation sont précisées avant de décrire le protocole expérimental et les résultats obtenus. Des annexes complètent l'exposé avec notamment le détail de l'implémentation technique de KISS et la conduite de l'expérimentation.

SOMMAIRE

1 Introduction

1.1 Rappel des objectifs de la tâche 6 : expérimenter et évaluer

La tâche 6, « Expérimentation et évaluation », a pour objectif d'évaluer, par l'expérimentation, les composants techniques du projet développés dans les tâches 2, 3, 4 et intégrés dans la tâche 5, à partir des deux scénarios de la tâche 1 : le scénario industriel et le scénario prospectif. La tâche 6.2 concerne l'expérimentation et l'évaluation de la production technique de CONTINUUM appliquée au scénario prospectif. Une description complète du scénario prospectif est rappelée en Annexe 1. Ce scénario, on le rappelle, a été validé par une étude amont de terrain effectuée auprès de 17 familles de la région grenobloise. Il s'agit donc d'un scénario réaliste. La méthode et les résultats sont consignés dans le document D4.1-4.2.

Le présent document (D6.2) a trait aux résultats de la tâche 6.2 avec l'évaluation de l'acceptabilité de KISS (Knit your Ideas into Smart Spaces), seul composant de l'infrastructure d'exécution de CONTINUUM, qui soit « visible » de l'utilisateur final. KISS est un outil qui permet à l'utilisateur final de « programmer » son habitat et d'en contrôler le comportement en fonction des changements de contexte. KISS a été appliqué à une séquence clef réaliste du scénario prospectif : le lever matinal et ses activités routinières. Il a été évalué dans DOMUS, l'appartement intelligent du Laboratoire d'Informatique de Grenoble (LIG), avec l'intervention de 13 sujets représentatifs.

1.2 Structure du document

Ce document est structuré comme suit :

- La section 2 qui suit rappelle les motivations qui nous ont conduits à développer KISS, puis consigne quelques définitions en lien avec KISS, de même le positionnement de KISS dans l'architecture Continuum.
- En 3, nous décrivons KISS du point de vue de l'utilisateur final, le détail de l'implémentation technique pouvant être consulté en Annexe2.
- La section 4 précise les adaptations nécessaires à la mise en œuvre de l'expérimentation tandis que la section 5 décrit le protocole expérimental adopté.
- Les résultats sur l'acceptabilité de KISS font l'objet de la section 6.
- Nous concluons avec les leçons qu'il convient de tirer de cette évaluation.

2 Rappels

Ces rappels portent sur 3 points : (1) nos motivations et approche retenue, (2) les définitions clefs nécessaires à la compréhension de l'exposé, et (3) l'intégration de KISS comme composant de l'infrastructure CONTINUUM.

2.1 Motivations et approche

Dans la pratique actuelle du développement de systèmes, la conception est assurée par des spécialistes (les designers), la réalisation revient à d'autres spécialistes (les informaticiens) et l'utilisateur est assimilé à une représentation archétypique réductrice (cf. le concept de persona [Preece 02]). Il en résulte une double dichotomie : d'une part, la séparation entre phase de conception et phase d'exécution, d'autre part, la distinction entre développeurs et utilisateurs. Au final, l'utilisateur, en bout de chaîne du processus de production, est non seulement un consommateur contraint par un système pensé et réalisé par d'autres, mais aussi une entité exclue de la boucle d'adaptation du système. Toutefois, à la décharge des concepteurs, l'utilisateur ne sait pas toujours exprimer ses besoins et encore moins en prévoir les évolutions.

Dans CONTINUUM, nous prenons le contre-pied de cette pratique en proposant de redonner le pouvoir à l'utilisateur. Dans cet esprit, l'utilisateur doit pouvoir tenir les rôles de concepteur et de développeur pour construire à façon des espaces intelligents capables de s'adapter à des besoins évolutifs. L'approche adoptée dans CONTINUUM est la suivante :

- Définition d'une infrastructure d'exécution orientée-services à trois niveaux d'adaptation : le niveau réflexe pour l'adaptation à des changements de contexte connus et prévus, le niveau tactique pour l'adaptation à des changements de contexte prévisibles, et le niveau stratégique pour l'adaptation à des changements de contexte inconnus (voir livrable D2.1) ;
- Définition, au sein de l'infrastructure d'exécution, de points de contrôle ouverts sur l'utilisateur appelés méta-IHM, à raison d'une méta-IHM par niveau d'adaptation (voir D4.1-4.2) : méta-IHM réflexe, méta-IHM tactique et méta-IHM stratégique. KISS, objet de ce rapport, est une méta-IHM tactique.

Nous rappelons ci-dessous les définitions nécessaires à la bonne compréhension de ce document.

2.2 Définitions : utilisateur final, espace et habitat intelligents, méta-IHM et programmation/développement/génie logiciel par l'utilisateur final

L'expression *utilisateur final* vient des économistes qui distinguent l'acheteur d'un produit, de l'utilisateur qui s'en sert au final.

Dans le contexte de CONTINUUM, l'expression « utilisateur final » permet de différencier l'informaticien professionnel, développeur de logiciels destinés au « marché », de la personne qui utilise *in fine* ce logiciel. Dans le cas du scénario prospectif, l'utilisateur final représentatif est membre d'une famille avec ses activités de la vie quotidienne.

Un *espace (ou environnement) intelligent* est un lieu (physique ou virtuel) d'activités (d'origine humaine ou non) constitué de dispositifs et de fonctions interconnectés, capable de garantir en toute circonstance les services attendus avec la valeur attendue.

Expliquons les termes clefs de cette définition.

- *Intelligent* implique adéquation, adaptation et respect de la valeur. Il n'implique pas, sans toutefois l'exclure, l'utilisation des paradigmes de l'Intelligence Artificielle pour la mise en œuvre technique.
- La *valeur* peut s'exprimer de manière réductrice en termes de qualité (au sens du Génie Logiciel). Mais comme le fait remarquer Cockton [Cockton 04, Cockton 05] puis Calvary [Calvary 07], il s'agit d'aller au-delà de métriques avec la prise en compte notamment de la satisfaction, du contentement, du plaisir, de toute sensation relevant du bien-être.
- Un *service* désigne une assistance ou commodité immatérielle fournie par l'espace intelligent. Il n'implique pas, sans toutefois l'exclure, l'utilisation du paradigme "Service-Oriented Architecture" pour la mise en œuvre technique.
- L'espace intelligent n'est *pas nécessairement un lieu clos* (l'activité en question peut être menée dans la rue, en pleine nature – c'est le cas du scénario industriel) ; il n'est *pas nécessairement physique* (l'activité peut avoir lieu dans un monde virtuel comme dans Second Life[1]). En vérité, il est une réalité mixte où le physique et le numérique se fondent en une "nouvelle réalité".
- Dès lors, les *dispositifs* auxquels nous faisons référence sont des artefacts (c.-à-d. des créations humaines) pouvant jouir d'*une double existence* – dans le monde physique et/ou dans un monde dual numérique.
- Notre définition assimile un espace intelligent à un lieu d'activités. En cela, nous reprenons à notre compte l'analyse de Harrison et de Dourish [Harrison 96] qui écrivent ceci : "Nous sommes situés dans un espace, mais nous agissons en des lieux. En outre, un lieu est un espace auquel nous attachons une valeur. C'est ce qui fait la différence entre une maison et un domicile. La maison nous protège du vent et de la pluie, mais le domicile, c'est l'endroit où nous vivons."[2]. Alors qu'Harrison et Dourish font référence à l'espace physique uniquement, nous étendons leur analyse aux espaces numériques.

Un *habitat intelligent* est un lieu d'activités (d'origine humaine ou non) constitué de dispositifs et de fonctions interconnectés, capable de garantir en toute circonstance les services attendus par ses habitants avec la valeur attendue comme le confort, l'économie d'énergie, la sécurité des biens et des personnes et plus généralement, le bien-être.

Nous convenons donc qu'un habitat intelligent, sur lequel porte le scénario prospectif, est un cas particulier d'espace intelligent.

[1] http://secondlife.com/

[2] "We are *located* in "space", but we *act* in "place". Furthermore, "places" are spaces that are valued. The distinction is rather like that between a "house" and a "home"; a house might keep out the wind and the rain, but a home is where we live. "

Une *méta-IHM* regroupe l'ensemble des fonctions (avec leur interface utilisateur) nécessaires et suffisantes pour évaluer et contrôler non seulement l'état mais aussi le comportement de systèmes intelligents et notamment leur processus d'adaptation [Coutaz 06].

Cet ensemble est méta- parce qu'il sert d'arche à tout l'espace intelligent. Il donne à l'utilisateur final les poignées pour intervenir aussi bien sur le comportement fonctionnel de l'espace intelligent que sur son comportement interactionnel. En somme, une méta-IHM est aux espaces intelligents, ce que le *desktop* et le *shell* d'Unix sont aux systèmes centralisés usuels. Un état de l'art et une synthèse sur ce sujet sont disponibles dans D4.1-4.2. Ainsi, KISS, en tant que point de contrôle humain des mécanismes d'adaptation de CONTINUUM, est une méta-IHM.

La « programmation par l'utilisateur final » (PUF) correspond à la situation où les rôles de développeur et de consommateur de programmes sont assurés par la même personne. La motivation du « programmeur-utilisateur final » est de produire un programme censé résoudre de manière automatisée un problème personnel qui serait plus coûteux à résoudre sans l'aide de la machine. Son objectif premier n'est pas d'apprendre à programmer mais de résoudre ce problème au moyen d'une machine de traitement de l'information [Nardi 95].

Dans le contexte de CONTINUUM, la motivation de l'utilisateur final est d'utiliser les services de l'habitat intelligent pour résoudre de manière automatisée un problème personnel qui serait plus coûteux à résoudre sans l'aide de ces services. Mais son objectif premier n'est pas, *a priori*, de devenir un spécialiste en habitat intelligent. Notre définition rejoint le point de vue de Ko et al. qui définissent la programmation par l'utilisateur final par son intention [Ko 11], non pas par le noviciat ou l'incompétence supposés de cet utilisateur en matière de programmation[3]. Blackwell, avec son modèle de l'investissement attentionnel (en anglais, « attention investment model »), donne corps à cette intention [Blackwell 02]. En effet, ce modèle structure le processus mental qui conduit une personne à s'impliquer dans une activité de programmation (ou à y renoncer) en fonction des coûts cognitifs à engager comparés aux bénéfices attendus et aux risques potentiels d'un tel programme. Dans le but de baisser ces coûts d'entrée, les chercheurs ont conçu des langages et des techniques de programmation dédiés (voir analyse dans D4.1-4.2). Autrement dit, programmer n'implique pas sans toutefois l'exclure, l'utilisation d'un langage de programmation pour professionnel.

Alors que l'on convient d'associer le terme PUF à la phase de création pure d'un programme, le « Développement par l'Utilisateur Final » (DUF) désigne « l'ensemble des méthodes, des techniques et des outils qui permettent, à un moment donné, à des utilisateurs de systèmes logiciels agissant en tant que développeurs …, de créer, de modifier, ou d'étendre un artefact logiciel.[4] » [Lieberman 06].

En effet, l'utilisateur final rencontre les mêmes problèmes que le développeur professionnel quand bien même, à l'heure actuelle, ses exigences en matière de processus et de qualité sont, au mieux, implicites. En effet, les développements par l'utilisateur final sont opportunistes et prospectifs [Brandt 08]. Le processus et les requis émergent avec l'expérience, au cours du développement. D'ailleurs, telle était la pratique à l'époque de l'informatique naissante. Mais la tendance au partage

[3] « We then define *end-user programming as programming to achieve the result of a program primarily for personal, rather than public us*e » [Ko 11, p. 21]

[4] Traduction de : EUD is « a set of methods, techniques and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact ».

et à la co-construction de solutions via les réseaux sociaux [Reppenning 11], conduira tôt ou tard à l'expression de requis extra fonctionnels. Si aujourd'hui personne n'est responsable en cas de mauvais fonctionnement [Kierkegaard 11], la question se posera inévitablement un jour ou l'autre. On trouvera dans [Ko 11] des exemples d'outils qui, bien que limités, conduisent les utilisateurs finaux à faire de la qualité malgré eux.

« Le Génie Logiciel par l'Utilisateur Final » (GLUF) vise à offrir les méthodes, les techniques et les outils de développement qui permettent d'assurer la production de solutions satisfaisant un ensemble de propriétés, ces propriétés définissant à leur tour la qualité requise de ces solutions. Autrement dit, GLUF = DUF + qualité[5]. En outre, le développement, qu'il soit de qualité ou pas, est susceptible d'être une entreprise collective. On parle alors de PUF, de DUF ou de GLUF social, voire citoyen.

En synthèse, KISS (Knit your Ideas into Smart Spaces) est un composant de l'infrastructure CONTINUUM qui, en tant que point de contrôle ouvert sur l'utilisateur final, joue le rôle de méta-IHM dont la couverture fonctionnelle relève de celle d'un outil de Développement par l'Utilisateur Final (DUF). KISS permet, en effet, de rédiger des programmes en langue pseudo-naturelle et de les mettre au point afin de définir et de contrôler le comportement d'espaces intelligents. En l'état, KISS n'inclut pas d'outils d'assurance qualité, ni ne supporte le co-développement. Dans ce document, nous rapportons l'évaluation de KISS appliqué au contrôle d'habitat intelligent. Avant cela, rappelons sa situation dans l'infrastructure CONTINUUM.

2.3 KISS dans l'infrastructure d'exécution CONTINUUM

L'infrastructure d'exécution CONTINUUM est un ensemble de services qui tirent parti des mécanismes d'adaptation dynamique de l'intergiciel WCOMP pour assurer la continuité de service dans les espaces intelligents.

2.3.1 Présentation générale

La figure 1 donne une représentation architecturale globale de l'infrastructure CONTINUUM :

- Les *services UPnP* de l'espace intelligent (en bas de la figure) encapsulent les dispositifs (équipements/objets) communicants de l'espace (lampes, interrupteurs, réveils, etc.).
- L'*application* est un service composite WCOMP qui encapsule l'assemblage des composants applicatifs en cours d'exécution.
- Deux composants particuliers appelés *designers*, assurent les fonctions suivantes : l'*AA Designer* calcule et effectue les (re)configurations de l'assemblage de l'application ; l'*UPnP Designer* fait le pont entre la notion de composant WCOMP et les services UPnP de l'environnement physique sous forme de *composants proxy* : à tout service UPnP présent dans l'environnement physique correspond un composant proxy dans l'application.
- En haut à droite, une base des composants disponibles sur étagère regroupe les composants susceptibles d'être recrutés dynamiquement par l'AA Designer. L'hypothèse implicite est que les métadonnées de ces composants sont exprimées dans une ontologie unique.

[5] "End-user programming focuses mainly on how to allow end users to create their own programs, and end-user software engineering considers how to support the entire software lifecycle and its attendant issues." [Ko 11]

À ces services de base, viennent se greffer trois unités fonctionnelles que nous détaillons ci-dessous : la Base de Connaissances (BdC), le Gestionnaire de Contexte (GC) et trois méta-IHM : méta-IHM-tactique-GC, méta-IHM-tactique-BdC et méta-IHM réflexe. En l'état, le niveau stratégique n'étant pas offert, il n'y a pas lieu d'offrir une méta-IHM stratégique.

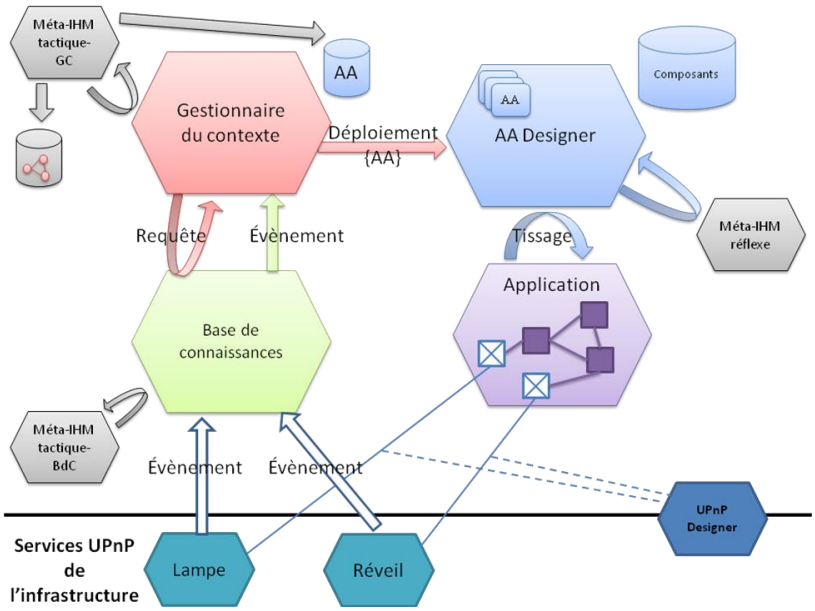


Figure 1. Représentation de l'architecture de CONTINUUM. Les hexagones représentent des services, les flèches des échanges entre services.

2.3.2 Base de Connaissances (BdC)

La *Base de Connaissances CONTINUUM* (BdC) maintient un modèle déclaratif de l'espace intelligent. Elle s'appuie pour cela sur les langages RDFS et SPARQL, deux standards du W3C, le premier pour déclarer des faits, le second pour interroger la BdC sur l'existence de faits. Ce choix technique offre l'avantage d'exploiter les capacités de raisonnement et d'inférences du Web sémantique. Les événements de modification de la base de faits (par exemple, faisant suite à l'arrivée ou au départ d'un dispositif UPnP, lampe ou réveil), de même les requêtes, typiquement « existe-t-il un dispositif d'affichage ? », se font via le protocole UPnP. Comme le montre la figure 1, la BdC reçoit des événements UPnP en provenance des dispositifs UPnP de l'espace intelligent. Le traitement de ces événements a deux effets : (1) modification éventuelle de la base de faits, (2) envoi au Gestionnaire de Contexte (GC) de deux types d'événements : événements de haut niveau correspondant aux modifications de la base de faits, et événements originaux des dispositifs UPnP si le GC en a exprimé l'intérêt. Enfin, la BdC répond aux requêtes issues notamment du Gestionnaire de Contexte (mais aussi de KISS, comme nous le verrons plus loin).

2.3.3 Gestionnaire de Contexte (GC)

Le *Gestionnaire de Contexte CONTINUUM* (GC) s'appuie sur les concepts introduits par [Rey 05 ; Coutaz 05] : un contexte est un graphe de situations qui partagent les mêmes ensembles de prédicats et d'entités. Dans CONTINUUM, ces entités sont les classes et instances de classes modélisées dans la BdC (Personne, Dispositif, etc.) et les prédicats sont des prédicats exprimables dans les termes de la BdC. Au sein d'un contexte donné, il y a changement de situation sur changement de valeur de prédicat et/ou sur changement de la cardinalité de l'ensemble des entités, par exemple l'arrivée d'un nouveau dispositif dans l'espace intelligent. Il y a changement de contexte si la nature de ces

ensembles change (ce qui correspond, *grosso modo*, à un changement d'ontologie). Le GC a pour mission de détecter les changements de situation, d'identifier la nouvelle situation et de là, s'il en est besoin, de produire un plan d'adaptation correspondant à la nouvelle situation et de le faire exécuter.

Pour cela, le GC interroge la BdC (requête SPARQL) sur réception de notification de changement d'état en provenance de la BdC notamment. À chaque situation correspond des AAs mémorisé dans une base d'AAs (en haut au centre de la figure 1) . Sur identification d'une nouvelle situation, le GC retire de l'AA Designer les AAs correspondant à l'ancienne situation, puis déploie sur l'AA Designer les AAs correspondant à la nouvelle. L'AA Designer, sollicité par ce nouveau déploiement, interprète les AAs déployés avec, comme conséquence, la (re)configuration du service composite Application. La base des plans d'adaptation « situation-AAs » n'est pas fixe. Elle évolue selon deux procédés complémentaires : d'une part, par un service d'apprentissage des comportements humains (non implémenté dans la version actuelle de l'infrastructure), d'autre part, via KISS qui joue le rôle de méta-IHM-tactique-GC.

2.3.4 Méta-IHM et niveaux d'adaptation

L'infrastructure d'exécution CONTINUUM introduit trois niveaux d'adaptation (réflexe, tactique et stratégique) par analogie avec les niveaux de compétences humaines avec, en regard, des exigences de latence spécifiques [Ferry 10] :

1. L'*adaptation réflexe* est assurée par l'AA Designer qui réinterprète les AA déployés sur apparition/disparition d'un composant proxy par l'UPnP Designer. Du point de vue de l'utilisateur final, la latence de l'adaptation réflexe doit être quasi-imperceptible (c.-à-d. inférieure à 80ms). À ce niveau réflexe, correspond une *méta-IHM réflexe* dont on a présenté un exemple dans D4.1-4.2[6].
2. L'*adaptation tactique* est assurée par le GC et la BdC qui, ensemble, permettent de détecter des changements de situations, d'identifier la nouvelle situation parmi celles qui sont connues, et de choisir dans la base d'AAs, celui qui convient. Du point de vue de l'utilisateur final, la latence de l'adaptation tactique est supposée être de l'ordre de la seconde. Nous convenons d'introduire *une méta-IHM-tactique* qui se décompose en deux parties : la *méta-IHM-tactique-GC* qui doit permettre à l'utilisateur final de programmer des AAs pour des situations prévisibles (ce sera le rôle de KISS) et la *méta-IHM-tactique-BdC* pour visualiser et modifier la BdC (non décrite ici, car utilisée par commodité par nous-mêmes en tant que développeurs de CONTINUUM).
3. L'*adaptation stratégique* est assurée par un service d'apprentissage non implémenté dans la version actuelle de CONTINUUM. Une *méta-IHM-stratégique* servirait alors à guider le service d'apprentissage en cas de doute et plus généralement à vérifier et corriger les faits appris.

Dans les sections qui suivent, nous décrivons KISS du point de vue de l'utilisateur final, puis les conditions expérimentales et ses résultats. On trouvera en Annexe 2, la description de la mise en œuvre technique de KISS.

[6] Par des gestes captés par une Kinect, l'utilisateur ordonne au système de redistribuer des composants de l'IHM d'un visualisateur de photos entre une table interactive, un mur et un smartphone (séquence « Au restaurant du scénario prospectif en Annexe 1). Sur le plan logiciel, cette redistribution implique la reconfiguration dynamique des composants de l'application. http://iihm.imag.fr/demo/photobrowser/

3 KISS vu de l'utilisateur final

Nous présentons d'abord KISS dans ses grandes lignes (3.1), puis nous en détaillons l'utilisation par un utilisateur final (3.2) et concluons en 3.3 par une synthèse sur KISS et son positionnement dans l'état de l'art.

3.1 Caractéristiques générales

KISS (Knit your Ideas into Smart Spaces) est un outil DUF qui se caractérise comme suit : il s'adresse à un utilisateur final constructeur. Sa couverture fonctionnelle inclut la programmation, la mise au point et dans une moindre mesure, la réutilisation. La mise au point peut se pratiquer au choix, dans le monde physique, dans un monde dual numérique ou dans les deux simultanément[7]. KISS ne fournit aucun support au co-développement, jugé pour l'instant, peu prioritaire. Par contre, il fait partie des rares exemples de l'état de l'art où phases de conception et d'exécution peuvent être perçues comme confondues.

Concernant la programmation, nous avons opté pour les choix suivants :

- Une *adaptabilité* de l'espace intelligent *par programmation*, soit *ex nihilo*, soit *à partir d'exemples réutilisables*. Il s'agit de *codage par spécification* avec *génération classique* mais, ce qui est plus original, le code généré est des plans d'adaptation AAs (cf. section 2.3). Ces plans, maintenus dans la base d'AAs de l'infrastructure d'exécution, bien qu'*accessibles à l'humain*, sont compréhensibles pour le développeur professionnel, mais ne sont pas destinés à l'utilisateur final.
- Un langage de programmation *déclaratif orienté règles* pour sa simplicité d'accès ainsi que l'ont démontrée des études sérieuses d'utilisabilité pour CAMP [Truong 04] et iCAP [Dey 06].
- Un *langage spécifique* dont la syntaxe et le lexique sont établis dynamiquement à partir de la BdC assurant *de facto* l'*extensibilité* et une *totale correspondance* avec le lieu de vie de l'utilisateur final.
- Une technique d'interaction WIMP[8] fondée sur l'utilisation de menus qui assure la *découverte progressive* du langage, impliquant un faible coût d'entrée puisque les unités du langage sont directement observables. Les options des menus, calculées dynamiquement à partir de la BdC, assurent du même coup un *support à la correction* : grâce aux menus fondés sur la BdC, qui est le reflet permanent de l'état de l'espace intelligent, l'utilisateur ne peut que construire des phrases sémantiquement correctes.
- Une *approche multisyntaxe avec égale opportunité* incluant une syntaxe de langage pseudonaturel (langue naturelle contrainte) et une syntaxe de langage visuel. Le langage visuel ayant fait l'objet d'une implémentation partielle, l'égale opportunité n'est dans les faits que *partielle*.
- Le choix d'un langage pseudonaturel (LPN) et la construction de phrases LPN au moyen de menus se justifient ainsi : d'après une étude portant sur la composition de services [Gabillon

[7] S'il est possible, par conception, d'exécuter un programme dans les deux mondes à la fois, la version actuelle de KISS ne permet l'exécution que dans un seul monde à la fois.
[8] WIMP pour « Window Icon Menu Pointer »

11], les utilisateurs préfèrent exprimer ces compositions en langue naturelle, en conformité avec la vie courante. Toutefois, la langue naturelle souffre de deux limitations : elle est ambiguë, exigeant des micro dialogues de clarification, et ne permet pas le « feedforward », c'est-à-dire de rendre observable l'espace du possible pour le discours à venir.

En réponse à la première difficulté, TellMe propose un mécanisme de négociation [Gil 11]. L'utilisateur saisit ses instructions dans un champ textuel et le système pose des questions si ces instructions sont ambiguës. Mais TellMe ne fournit pas de feedforward. En réponse à cette seconde difficulté, nous avons repris à notre compte une solution des années 80 à base de menus contextuels qui permettent de construire des phrases correctes en LPN [Tennant 83].

3.2 KISS en action

La tâche centrale de l'utilisateur final consiste, rappelons-le, à programmer l'espace intelligent dans lequel il vit, et notamment à définir le comportement de son domicile au quotidien. Du point de vue de cet utilisateur, KISS se présente donc comme un éditeur de programmes et d'un metteur au point. La figure 2 montre l'organisation concrète de KISS pour notre expérimentation.

Figure 2. KISS expérimental. Sur la table de gauche, le laptop sur lequel s'exécute l'éditeur de programmes. Sur la table de droite, le laptop du metteur au point et sa tablette pour jouer sur le temps.

3.2.1 L'éditeur de KISS

La figure 3 présente la vue générale de l'éditeur : au sommet, la phrase en cours d'édition, en bas, la liste des phrases déjà construites. En bas à gauche, un bouton pour créer une nouvelle phrase. Les figures qui suivent illustrent un scénario de construction de phrases en LPN.

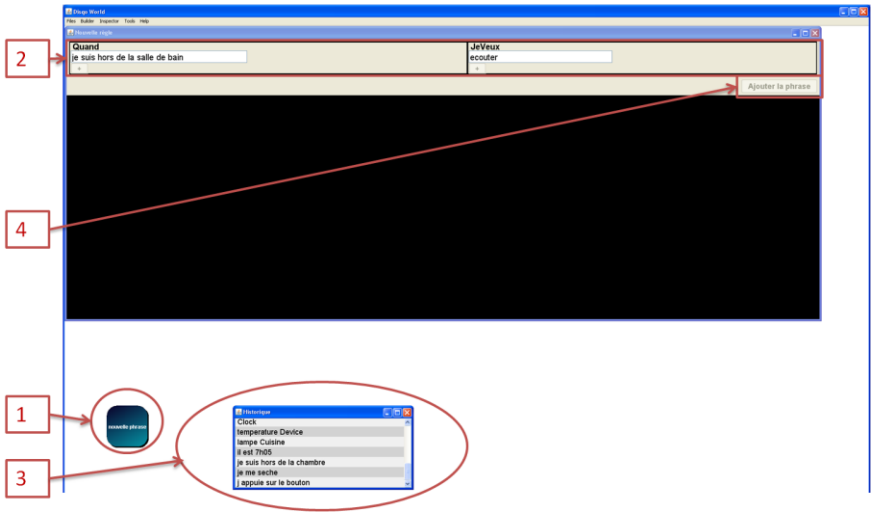


Figure 3. Vue d'ensemble de l'éditeur KISS. En 1, le bouton qui permet, en le cliquant, de rédiger une nouvelle phrase en langage LPN. En 2, la zone d'édition de la phrase LPN courante, en l'occurrence : *Quand je suis dans la salle de bain, je veux écouter …*. En 3, la liste des phrases déjà écrites, repérées chacune par un titre. En 4, le bouton pour sauvegarder la phrase courante.

Le paradigme de programmation, nous l'avons vu, est de type déclaratif par règle. La zone d'édition comprend donc une partie condition intitulée *Quand* et une partie action, *Je veux*. Chaque partie est à son tour constituée de champs qui, une fois remplis (au moyen de menus déroulants) peuvent faire apparaître d'autres champs. La figure 4 en donne une illustration.

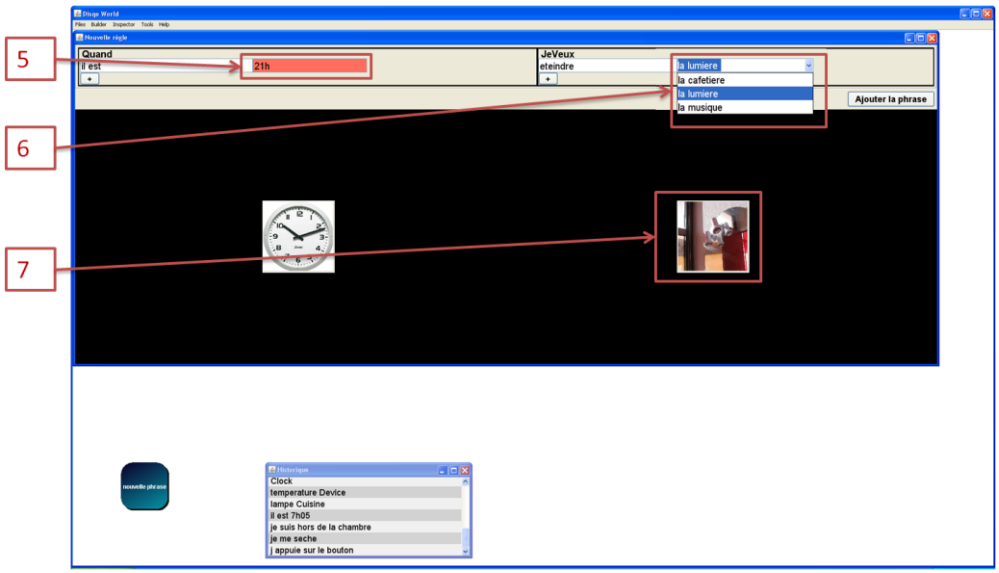


Figure 4. Construction d'une phrase LPN avec KISS. Pour remplir un champ donné, il suffit de placer la souris sur ce champ : un menu apparaît offrant les possibilités sémantiquement correctes parmi lesquelles l'utilisateur fait son choix. Dans l'exemple, l'élément *lumière* va être retenu comme paramètre de l'action éteindre. Le choix d'un élément de menu peut entraîner l'apparition de nouveaux arguments (champs) dont il faut spécifier la valeur. Ceux-ci sont généralement coloriés en rouge comme en 5. Lorsqu'un champ fait référence à un équipement de l'habitat, son image est affichée comme en 7 (ici, l'image de la lumière qui vient d'être choisie). La phrase actuelle est maintenant : *Quand il est 21h, je veux éteindre la lumière*.

L'utilisateur a la possibilité d'affiner les parties conditions et actions par ajout ou retrait de conditions et d'actions. La figure 5 montre comment. Une fois la phrase complète, il est possible de la mémoriser et de lui donner un nom. Ce nom apparaît alors dans la liste, dite *historique*, des phrases rédigées (en bas de l'écran). Toute phrase de l'historique peut être sélectionnée et éditée. KISS supporte ainsi une réutilisation d'exemples *a minima*.

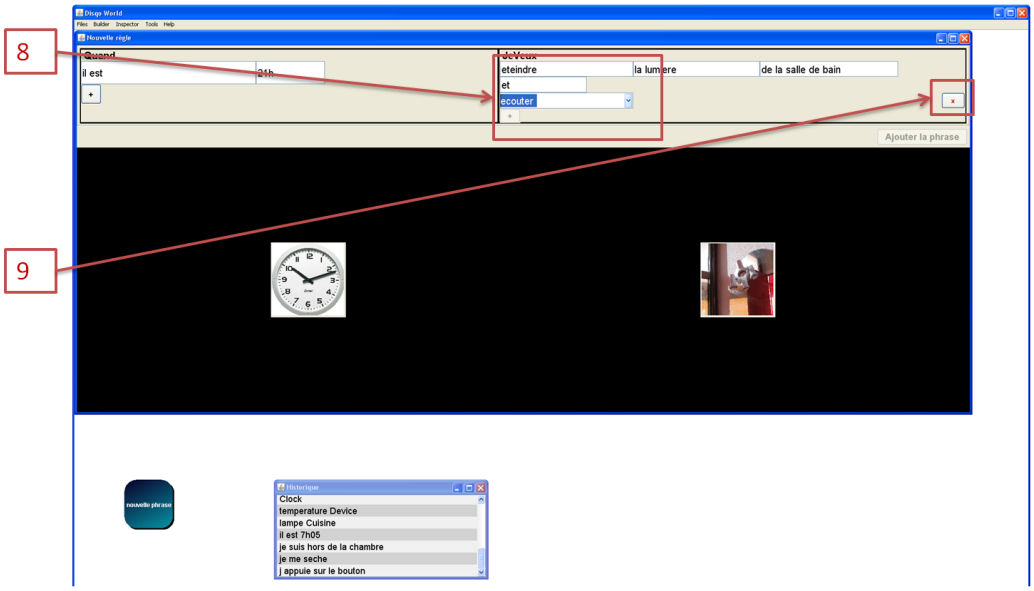


Figure 5. Ajout et retrait de conditions et d'actions. La sélection du bouton « + » (comme en 8) en-dessous de la portion de phrase rédigée fait apparaître un menu dans lequel il suffit de sélectionner l'opérateur logique souhaité. Les boutons « x » (comme en 9) situé en regard de chaque action et condition permettent de supprimer les actions et conditions correspondantes. La phrase actuelle est maintenant : *Quand il est 21h, je veux éteindre la lumière de la salle de bain et écouter*. Ecouter exigeant un argument, un nouveau champ va être présenté.

La sauvegarde d'une phrase entraîne sa traduction en un plan d'adaptation « situation-AAs » où la situation vient de la partie *Quand* de la phrase et l'ensemble des AAs à déployer lorsque la situation se présentera, est issu de la partie *Je veux*. Le détail de cette traduction est fourni dans l'annexe 2.

3.2.2 Le metteur au point de KISS

Le metteur au point de KISS permet d'observer l'effet d'une phrase soit dans l'environnement physique réel, soit dans une représentation duale virtuelle dit simulateur (soit dans les deux à la fois). Le simulateur est nécessaire au test de situations qui ne correspondent pas à la situation présente de la vie réelle, par exemple l'existence d'une fuite de gaz ou un événement qui se produira plus tard dans le temps. La figure 6 montre le simulateur en représentation graphique 2D. Une représentation 3D, plus réaliste mais jugée moins utilisable, a également été produite (figure 7). La figure 8 montre l'IHM présentée sur la tablette graphique pour régler l'heure et décider du lieu d'exécution du programme (c.-à-d. dans le monde réel ou dans le monde virtuel).

Prenons le cas du test de la phrase *quand il est 7h du matin et que je rentre dans la salle de bain, la lumière s'allume et la cafetière se met en marche,* et supposons que dans le monde réel, au moment de la rédaction de cette phrase, il est 14h00. L'utilisateur ajuste l'heure dans le champ 2 de la figure 8 (afin de ne pas attendre le lendemain matin !) et choisit le mode d'exécution de la phrase (simulateur ou appartement). Ensuite, il déplace son avatar dans la salle de bain. En mode *simulateur*, le résultat de l'exécution de la phrase se constate dans la représentation graphique : la lumière de la salle de bain est maintenant allumée et la cafetière est en marche. En mode *appartement*, les actions sont appliquées à la cafetière réelle et à lumière de la salle de bain physique. Le détail du fonctionnement du metteur au point est fourni dans l'Annexe 2.

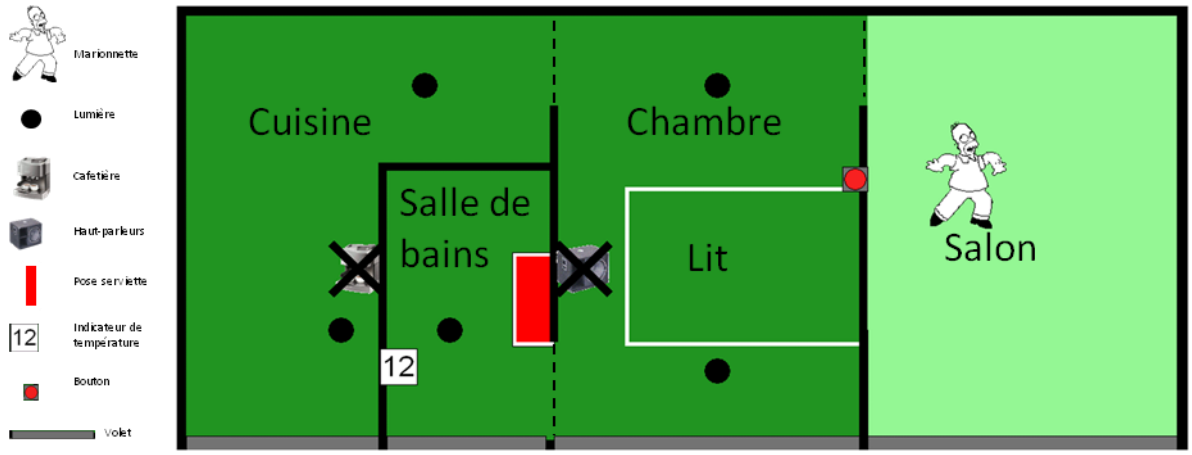


Figure 6. Le simulateur, représentation graphique 2D du monde réel (ici l'appartement utilisé pour notre expérimentation). On y voit les différentes pièces de l'appartement, l'état du mobilier instrumenté et la localisation de l'utilisateur : lit (augmenté de capteur de présence), volets (ouverts, semi-ouverts ou fermés), lumières (allumées ou éteintes), haut-parleurs (diffusant ou non de la musique), cafetière (en fonction ou éteinte), indicateur de température dans la salle de bain. L'utilisateur est représenté par la marionnette Homer déplaçable par glisser-déposer pour reproduire sans avoir à se déplacer dans le monde réel, des conditions de déplacement ou de localisation comme dans *quand je rentre dans la salle de bain…* ou encore *quand je suis dans mon lit…*

Figure 7. Le simulateur en représentation graphique 3D selon une vue à la 3ème personne.

Figure 8. Spécification de l'heure et du lieu de test du metteur au point de KISS. En 2, un champ permet de spécifier l'heure actuelle (pour se placer dans l'avenir par exemple). En 3, la sélection des flèches permet d'agir sur le coefficient d'accélération du temps (valeur courante affichée en 1) afin que l'utilisateur en situation de test n'ait pas à subir la durée du monde réel. En 4, les deux boutons pour choisir le lieu d'exécution du programme (dans le simulateur ou dans l'appartement).

3.3 Synthèse et positionnement de KISS dans l'état de l'art

En synthèse, KISS est un outil DUF qui permet à l'utilisateur final de rédiger des programmes en langue française pseudonaturelle et d'en tester l'exécution dans le monde réel et/ou dans son dual virtuel. Sur le plan technique, KISS est l'un des services de l'infrastructure d'exécution CONTINUUM qui, à son tour, s'appuie sur l'intergiciel WCOMP pour la composition et la reconfiguration dynamiques. Composition et reconfiguration sont exprimées en Aspect d'Assemblage (AA). KISS, qui joue le rôle de méta-IHM tactique dans l'infrastructure CONTINUUM, traduit les programmes de l'utilisateur en plans d'adaptation sous la forme de couples « situation-AA ».

Par rapport à l'état de l'art en matière de DUF, l'originalité de KISS est d'intégrer simultanément tous les points clefs suivants :

- une approche multisyntaxe (avec égale opportunité partielle dans sa version actuelle) : l'utilisateur a le choix entre plusieurs syntaxes simultanément disponibles ;
- un langage LPN avec menus contextuels en complète conformité avec l'état de l'espace intelligent : l'utilisateur rédige nécessairement des phrases syntaxiquement et sémantiquement correctes ;
- une génération de code, non pas de code classique, mais de plans d'adaptation compréhensibles par une infrastructure à composants orientés service et de là :
- une fusion des phases de conception et d'exécution ;
- une exécution qui peut avoir lieu dans le monde réel et/ou dans un monde dual numérique simulé offrant une grande souplesse dans la mise au point.

Si l'état de l'art en matière de DUF est avancé pour des applications de bureautique (les tableurs en particulier), peu a été fait pour les espaces intelligents. En gros, nous trouvons deux types de propositions :

- Les outils émanant de la communauté IHM conçus selon une approche centrée utilisateur. Il s'agit de iCAP [Dey 06, Sohn 06], CAMP [Truong 04], AutoHAN [Blackwell01, Blackwell 04] et, dans une certaine mesure, de BYOB (Build Your Own Block) [Ash 11][9].
- Les outils émanant de la communauté systèmes répartis et intergiciels avec TeC [Sousa 11] et PANTAGRUEL [Drey 10].

Les premiers sont marqués par le souci de définir un outil conforme aux besoins de l'utilisateur final sans trop d'égard pour l'infrastructure d'exécution ; les seconds portent l'attention sur l'intégration de l'outil dans une infrastructure d'exécution puissante et efficace, sans trop d'égard pour l'utilisabilité. KISS intègre les deux aspects : centrage sur l'utilisateur (étude amont effectuée la première année du projet) et intégration comme point de contrôle dans l'infrastructure CONTINUUM/WCOMP.

Parmi les outils DUF représentatifs, seul AutoHan est multisyntaxe, mais il n'inclut pas de metteur au point. Comme KISS, la plupart adopte un paradigme de programmation déclaratif par règles, mais seul CAMP utilise comme KISS un langage pseudo-naturel, sans toutefois offrir de metteur au point. Dans ce paysage, BYOP, une extension du langage impératif visuel SCRATCH [Resnik 09] pour la programmation d'appareils domestiques, fait exception. Toutefois, les développements avec BYOB sont des applications jouets, l'infrastructure d'exécution ne passant pas à l'échelle.

Avec TeC et PANTAGRUEL, la programmation est clairement influencée par le modèle à composants dynamiques de l'infrastructure d'exécution et de déploiement alors que dans KISS ce modèle est caché derrière un langage LPN. Dans TeC, l'utilisateur spécifie explicitement les connexions entre les composants et PANTAGRUEL construit ces connexions à partir de la spécification de règles d'orchestration entre objets (capteurs, services). TeC n'inclut pas d'aide à la mise au point, mais grâce à son infrastructure d'exécution dynamique, les phases de conception et d'exécution sont, comme dans KISS, fortement couplées alors qu'il y a séparation stricte dans PANTAGRUEL. Comme KISS, PANTAGRUEL inclut un metteur au point dans le monde numérique. Notons que tout programme PANTAGRUEL peut être traduit en TLA+ et de là, avec le model checker TLC, un spécialiste averti peut conduire une campagne de vérification formelle de propriétés clefs du système.

Ayant rappelé le contexte de cette étude, nous sommes maintenant en mesure de détailler l'évaluation de KISS du point de vue de l'utilisateur. Nous commençons par préciser les adaptations nécessaires à la réalisation des tests et les compromis réalisés entre la faisabilité technique dans le temps imparti et le respect du requis de qualité « écologique » (section 4). Puis, nous présentons le protocole expérimental (section 5) suivi, en section 6, de l'analyse des résultats.

[9] http://en.wikipedia.org/wiki/BYOB_(programming_language)

4 Préparatifs expérimentaux

Rappelons que WCOMP intègre le protocole UPnP et que l'évaluation de KISS s'est faite dans DOMUS, un appartement entièrement meublé. Comme le montre la figure 9, DOMUS est équipé de capteurs et de dispositifs communicants contrôlables. Certains de ces équipements sont natifs UPnP. C'est le cas par exemple des équipements audio. D'autres ont nécessité des ajustements pour les rendre compatibles UPnP.

Bien qu'il ait été conçu pour mener des expérimentations sur l'habitat intelligent, DOMUS n'est pas directement « Plug and Play ». Nous avons donc décliné les préparatifs expérimentaux en deux volets : (1) Vérification de la faisabilité de l'extrait de scénario dans DOMUS. (2) Vérification de l'intégration technique de KISS et de CONTINUUM/WCOMP dans DOMUS. Selon toute attente, il a fallu faire des adaptations à la fois du scénario et de la plate-forme d'accueil, motivées par un bon compromis entre pragmatisme et réalisme.

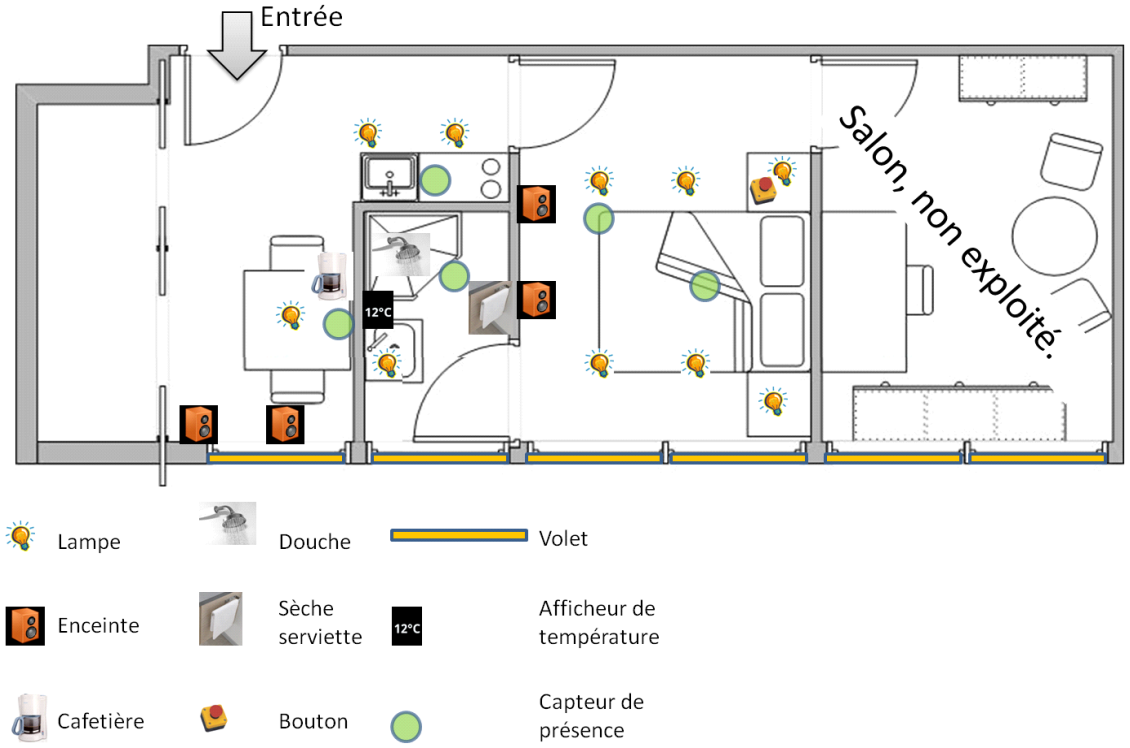


Figure 9. Plan de DOMUS avec implantation des différents équipements référencés dans notre test. Le salon n'est pas référencé dans notre protocole expérimental.

4.1 Influence du scénario

Le scénario original peut être consulté dans sa totalité dans l'Annexe 1. De façon à limiter la durée des passations tout en respectant le requis de réalisme, nous en avons extrait une séquence pertinente : le lever matinal qui relève d'activités routinières, activités auxquelles l'intelligence ambiante est susceptible d'apporter des améliorations [Coutaz 10, Davidoff 06]. Afin de nous assurer de la faisabilité de cette séquence dans DOMUS, nous avons vérifié le fonctionnement de chaque service et équipement référencés dans l'extrait de scénario. Ces vérifications systématiques ont appelé trois types d'actions correctives : l'adaptation de l'extrait de scénario, l'ajout d'équipements dans DOMUS et l'introduction d'un Magicien d'Oz.

4.1.1 Adaptation de l'extrait de scénario

Nous avons adapté le scénario du lever matinal en y apportant deux suppressions, une modification et un ajout.

1) L'extrait original fait référence aux toilettes : *il prend le chemin des toilettes*. DOMUS ne disposant pas de toilettes, nous avons dû supprimer cette référence.

2) L'extrait original fait référence à la télévision : *Allumer le téléviseur*. DOMUS possède un téléviseur dont le pilotage via UPnP était alors en cours d'implémentation. Nous avons supprimé la référence à cet équipement au regard du requis de robustesse.

3) L'extrait original fait référence à la radio de la salle de bain alors qu'il n'y en a pas dans celle de DOMUS. Or DOMUS est un petit appartement de 40m² environ aux cloisons fines. Il s'ensuit que le son joué dans la chambre s'entend tout aussi bien depuis la salle de bain. Nous avons modifié l'extrait du scénario en conséquence.

4) Comme DOMUS dispose de volets UPnP et que la séquence se déroule le matin au réveil, nous avons jugé pertinent d'ajouter l'ouverture des volets au lever du lit.

Enfin, l'extrait de scénario a été simplifié et reformulé de façon à ce que les utilisateurs participant au test d'acceptabilité puissent se l'approprier facilement. Le texte final de cet extrait est présenté dans la figure 10.

> Vous habitez dans cette maison intelligente qui contient des objets programmables : machine à café, volets, lumière, capteurs de présence, pose-serviettes, etc. Grâce à ces nouveaux dispositifs, vous allez pouvoir programmer votre maison, en particulier l'enchaînement des tâches du matin. Sur la table de chevet, un bouton vous permet de lancer la routine du matin. Le matin, vous avez l'habitude de vous réveiller à 7h. Vous aimez vous faire réveiller par une musique douce et une lumière tamisée. Si vous restez au lit trop longtemps, vous aimeriez que l'intensité de la lumière et de la musique augmente. Quand vous êtes décidé à vous lever, vous appuyez sur le bouton et vous vous levez. La lumière change d'intensité. Le volume sonore passe à un niveau acceptable. Une fois que vous êtes levé, vous souhaitez que les volets s'ouvrent. Ensuite, vous allez dans la salle de bain, la lumière s'allume dès que vous rentrez dans la pièce. L'eau de la douche est préchauffée et une lumière projetée sur la douche vous indique la température de l'eau. Quand vous sortez de la douche, et commencez à vous sécher, la cafetière se déclenche de sorte que le petit-déjeuner soit prêt.

Figure 10. Le scénario du *réveil matinal* soumis aux participants pour programmer DOMUS avec KISS et l'infrastructure CONTINUUM adaptée.

4.1.2 Ajout d'équipements dans DOMUS

L'extrait de scénario fait référence à un indicateur lumineux de la température de l'eau : *une lumière projetée sur le plafond au-dessus de la douche lui indique la température de l'eau*. Cette phrase nécessite la présence d'un capteur de température et d'un afficheur que nous avons intégrés dans DOMUS (figure 11). L'intérêt de cet ajout est de rendre possible l'expression de règles comprenant des verbes d'aspect duratif. Dans *je veux regarder la température de l'eau sur la douche*, le verbe d'action « regarder » est d'aspect duratif alors que les verbes utilisés dans l'extrait de scénario sont d'aspect inchoatif comme dans *allumer la lumière*.

Figure 11. L'afficheur de température réalisé avec un SmartPhone.

4.1.3 Utilisation de la technique du Magicien d'Oz

La technique du Magicien d'Oz consiste à faire simuler par un compère humain les fonctions manquantes d'un système de telle sorte que l'utilisateur croie que ces fonctions sont véritablement assurées par le système [Dahlbäck 92]. L'existence du compère, un expérimentateur averti, est inconnue de l'utilisateur avant et pendant les passations.

Dans notre extrait de scénario, certains comportements de l'habitat sont conditionnés par l'existence de capteurs dont DOMUS ne dispose pas, ou bien de capteurs dont la fiabilité est insuffisante. Par exemple, la lumière doit s'allumer quand on rentre dans une pièce et s'éteindre quand il n'y a personne. Il s'avère que les capteurs de DOMUS détectent le mouvement, non pas la présence. En conséquence, la présence d'une personne en mouvement est détectée et signalée immédiatement. En revanche, l'absence de cette personne, qui devient vraie lorsqu'elle sort de la pièce, est signalée par DOMUS au bout d'une trentaine de secondes. Ce délai est pénalisant. De même, le porte-serviettes, le lit et le bouton starter (qui sert à démarrer de la maison de manière globale), sont équipés de capteurs peu fiables ou bien ne sont pas instrumentés. Profitant de l'existence d'une régie, nous avons implémenté un service de Magicien d'Oz[10] (MOz) pour assurer correctement et de manière fiable les fonctions manquantes de DOMUS.

La régie du plateau d'expérimentation, qui inclut des caméras et des microphones d'observation, permet au compère de suivre en temps réel les activités menées dans DOMUS. Le compère, installé dans la salle de la régie, utilise MOz pour remplir deux rôles :

1. Simulation des capteurs : présence/absence de l'utilisateur dans une pièce, présence/absence de l'utilisateur sur le lit, utilisation du porte-serviettes, actionnement du bouton starter.
2. Simulation des prédicats de la Base de Connaissances. Nous reviendrons sur ce point dans la section qui suit à propos de l'infrastructure d'accueil.

La figure 12 montre l'IHM du service MOz correspondant à la simulation des capteurs. Cette IHM, avec la représentation 2D du DOMUS réel, reprend celle du metteur au point mis à la disposition de l'utilisateur final. Il est important de préciser que les capteurs virtuels sont vus sur le réseau comme des dispositifs UPnP au même titre que les capteurs réels. Par glisser-déposer, le compère, qui surveille les déplacements de l'utilisateur participant, place la marionnette Homer au bon endroit dans la représentation graphique du DOMUS virtuel. Par un clic souris, il désigne les représentations du porte-serviettes, du bouton de démarrage de la maison et du lit lorsque le participant les utilise dans le DOMUS réel.

[10] MOz est réalisé avec la boîte à outils COMETs [Demeure 08]

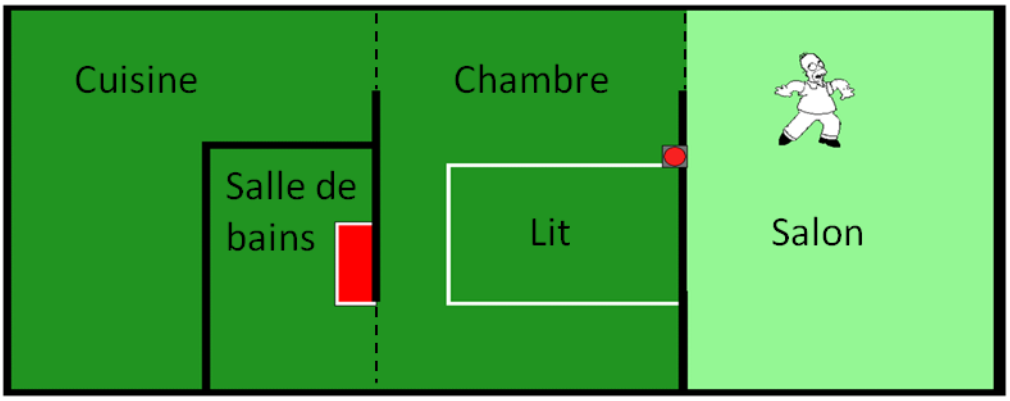


Figure 12. IHM du service MOz utilisée par le compère pour pallier les limitations des capteurs du DOMUS réel. Le compère déplace Homer (représentant de l'utilisateur participant) dans le DOMUS virtuel de manière synchronisée avec les déplacements de l'utilisateur dans le DOMUS réel. Une zone sur fond clair indique la présence actuelle simulée. Le bouton starter est représenté par le disque rouge à côté du lit ; le porte-serviettes de la salle de bain, par le rectangle rouge. Le compère désigne ces représentations par un clic souris lorsque le participant les utilise dans le DOMUS réel.

Nous venons de présenter les adaptations guidées par l'extrait de scénario pour, *in fine*, disposer d'une séquence réaliste et réalisable dans DOMUS. Nous considérons maintenant les adaptations techniques relatives à l'infrastructure d'accueil.

4.2 Influence de l'infrastructure d'accueil

Nous détaillons dans cette section comment nous avons concilié les capacités techniques de DOMUS et de CONTINUUM/WCOMP pour répondre à nos besoins pratiques d'intégration logicielle ainsi qu'à nos requis de latence et de robustesse. Ces adaptations se répartissent en quatre points : extension logicielle pour les équipements UPnP, remplacement de WCOMP, simulation de la Base de Connaissances et développement d'un second appartement virtuel.

4.2.1 Extension logicielle des équipements UPnP

Dans WCOMP, on le rappelle, tout équipement UPnP est représenté, dans le container applicatif, par un composant proxy. À chaque apparition ou disparition de proxy, l'AA Designer de WCOMP identifie les points de coupe pour ensuite reconfigurer les composants du container applicatif. La recherche des points de coupe s'opère sur les métadonnées des équipements (type de l'équipement, localisation, taille, etc.) pour lesquelles le protocole UPnP n'offre pas de standard de représentation.

En conséquence, WCOMP introduit son standard en imposant l'existence d'un service « métadonnées ». Nous avons dû implémenter ce service pour tous les équipements UPnP de DOMUS. Ce travail a été relativement aisé pour les équipements dont la partie UPnP avait été développée par l'équipe MultiCom du LIG (équipe qui gère DOMUS). Pour les équipements dont nous n'avions pas le code source (cas de l'audio notamment), nous avons dû développer des proxy enrichis du service « métadonnées ».

4.2.2 Remplacement de WCOMP par un intergiciel dédié

Les phrases de l'éditeur KISS sont traduites en Aspects d'Assemblages (AA). Si des tests concluants ont été réalisés en laboratoire sur un petit nombre de composants et d'AAs, il n'en a pas été de même avec le passage à l'échelle dans DOMUS. En effet, en tant que développeurs de KISS, nous avions besoin de vérifier la correction des AAs générés par KISS (par exemple, les liaisons entre les

composants). L'environnement WCOMP inclut un outil de mise au point, mais comme le montre la figure 13, son interface graphique ne se prête pas à la représentation de nombreux composants.

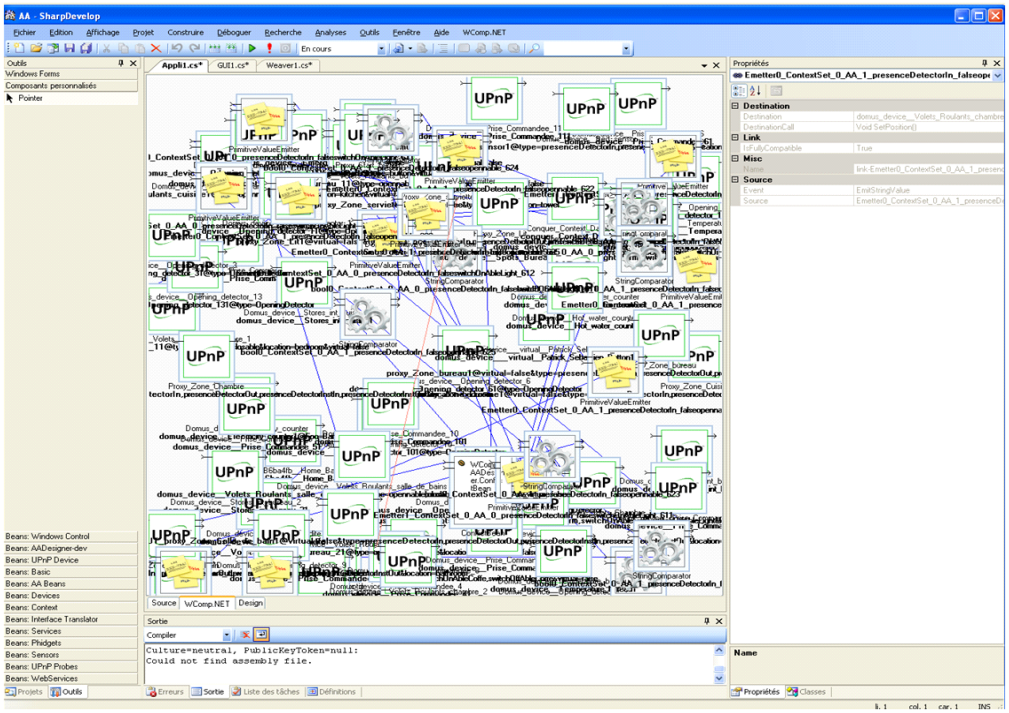


Figure 13. Etat de l'IHM de l'outil de mise au point de WCOMP après le tissage de composants correspondant à la phrase *Quand je rentre dans la chambre, fermer tous les volets et allumer la lumière de la chambre avec une intensité moyenne*.

Outre la difficulté de la mise au point, plusieurs problèmes sont apparus lors des pré-tests dans DOMUS :

- Détection incomplète des appareils UPnP lorsqu'ils existent en grand nombre (de l'ordre de la cinquantaine) et notamment des lumières indispensables au scénario expérimental. Etait-ce dû à une défaillance de la pile UPnP utilisée ?
- Difficultés avec WCOMP concernant l'affichage de multiples fenêtres d'erreur[11] (dû à des AA mal générés) jusqu'à quelques crashs de l'intergiciel. Il nous a été difficile, voire impossible en l'absence de support en interne, d'identifier l'origine de ces erreurs et donc d'y remédier rapidement.

Ces difficultés nous ont poussés à implémenter un intergiciel de remplacement dit « de secours » avec la boîte à outils COMETs [Demeure 08]. Cet intergiciel de secours, développé sur mesure en 330 lignes de Tcl, ne vise nullement à concurrencer WCOMP[12]. Il permet seulement :

- d'observer les composants UPnP présents sur le réseau et de détecter ceux qui sont dotés du service « métadonnées ».
- de gérer des listes d'AA. Comme dans WCOMP, ces AA ont une partie condition (qui porte sur la présence de composants) et une partie action (qui relie des composants).

L'intégration de l'intergiciel de secours a été facilitée par l'architecture de KISS dans laquelle les briques dépendantes de l'infrastructure d'accueil sont clairement identifiées. Ainsi, seul le

[11] Ce problème a été corrigé depuis.
[12] DOMUS avait été réservé pour une date précise : nous ne pouvions donc reculer la date des expérimentations.

générateur de langage cible a nécessité une réécriture pour remplacer les AA WCOMP par des AA « maison » (des scripts de reconfiguration) compris par l'intergiciel de secours.

4.2.3 Simulation de la Base de Connaissances

Une première version de la BdC a fait l'objet d'une intégration avec KISS, mais nous n'avons pas pu intégrer la dernière version de la BdC dont le développement technique évoluait constamment. Au vu des besoins de notre expérimentation, une simulation simplifiée de la BdC s'est révélé être un bon compromis.

En effet, notre extrait de scénario utilise un nombre limité de prédicats : d'une part, la séquence est courte et se déroule à un instant précis de la journée limitant *de facto* le nombre de situations et de contextes ; d'autre part, les prédicats que la BdC doit traiter ne portent que sur des conditions d'aspect duratif. Or, dans le scénario expérimental, la majorité des conditions est d'aspect inchoatif.

La BdC simulée (figure 14) constitue la deuxième fonction de MOz. L'IHM qui lui correspond présente son état sous la forme d'une liste de couples <case à cocher – prédicat>. Le compère coche ou décoche les cases en fonction des situations observées. Lorsque le Gestionnaire de Contexte (GC) envoie une demande d'évaluation d'un prédicat donné, la BdC retourne *Vrai* si la case est cochée, *Faux* dans le cas contraire.

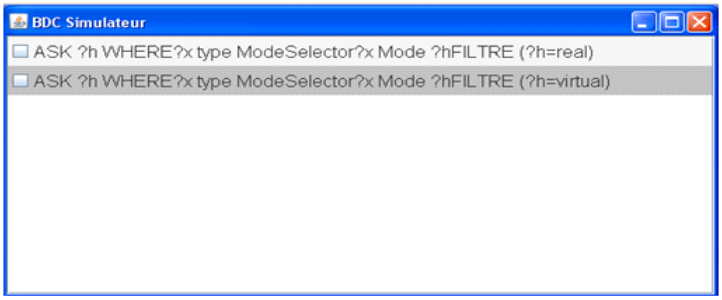


Figure 14. IHM du service MOz utilisée par le compère pour simuler l'évaluation de prédicats. Chaque ligne est un prédicat dont l'évaluation est demandée par le GdC. La réponse retournée par MOz dépend de l'état de la case à cocher correspondante.

En synthèse, le transfert des travaux de laboratoire vers une structure d'accueil de type intelligence ambiante comme DOMUS exige des adaptations inattendues qu'il faut savoir prévoir dans un plan de développement. Nous tirons de cette expérience des enseignements et des recommandations que nous synthétiserons en conclusion du rapport.

Ces différents ajustements aboutissent à un extrait de scénario programmable par l'utilisateur final et applicable dans DOMUS. La section suivante concerne le protocole expérimental retenu.

5 Protocole expérimental

L'objectif du test d'acceptabilité est double :

- D'une part, valider l'usage du langage pseudonaturel comme moyen de programmer le comportement d'un habitat intelligent. En particulier, il s'agit de vérifier que l'utilisateur peut exprimer aisément ses objectifs avec le vocabulaire fourni, qu'il comprend quels sont les objets sur lesquels porte le programme qu'il rédige et qu'il maîtrise la sémantique de ses programmes.
- D'autre part, valider l'utilité d'une réplique virtuelle de habitat réel pour la mise au point des programmes. En particulier, il s'agit de vérifier que l'utilisateur est capable de se repérer dans l'habitat virtuel et d'y retrouver les objets qu'il programme.

Nous synthétisons le déroulement du protocole expérimental, puis nous présentons le recrutement des participants. Le déroulement complet du test peut être consulté en Annexe 3.

5.1 Déroulement de l'expérimentation

Le protocole expérimental comprend six étapes : accueil, visite de l'appartement, formation à l'utilisation de KISS, exécution du scénario par le participant, vérification du bon déroulement du scénario, débriefing avec questionnaire.

1) Le participant est accueilli sur le site par l'expérimentateur qui lui présente l'expérimentation en quelques mots. Il lui est demandé de signer son accord de participer à l'expérimentation.

2) L'expérimentateur lui fait visiter l'appartement DOMUS. Il lui fournit un plan de DOMUS (celui de la figure 9) et lui montre pièce par pièce les différents équipements, y compris les capteurs. Une fois la visite terminée, l'expérimentateur s'assure que le participant a compris l'agencement de DOMUS et de ses équipements.

3) Le participant est formé à l'utilisation de KISS (éditeur de phrases, appartement virtuel et horloge). Pour cela, l'expérimentateur rédige une phrase en langage pseudonaturel (LPN) : *quand je suis dans la salle de bain, je veux allumer la lumière de la salle de bain avec une intensité moyenne*. Le participant est ensuite invité à se rendre dans la salle de bain pour constater l'éclairage. Quand il en sort, il s'aperçoit que la lumière reste allumée. L'expérimentateur en explique la raison et propose au participant de rédiger une phrase pour éteindre la lumière de la salle de bain quand il en sort. Le participant rédige la phrase correspondante et teste le résultat en entrant et sortant de la salle de bain.

L'expérimentateur propose ensuite d'ajouter une contrainte horaire à la phrase précédente. Il montre au participant comment revenir sur une phrase déjà rédigée puis le guide pour reformuler la phrase avec des contraintes horaires : *quand je suis dans la salle de bain et quand il est entre 20h et 22h, je veux allumer la lumière de la salle de bain avec une intensité moyenne*. Pour tester cette phrase, le participant doit changer l'heure, ce qui permet à l'expérimentateur de lui présenter l'horloge UPnP et de lui en expliquer le fonctionnement. Enfin, l'expérimentateur présente l'appartement simulé. Il explique qu'il s'agit d'une reproduction de DOMUS et y situe les pièces et équipements. L'expérimentateur propose au participant de déplacer l'avatar dans la salle de bain, de le sortir, puis de refaire la même manipulation en changeant l'heure. À la fin de cette étape,

l'expérimentateur s'assure que le participant a bien compris le fonctionnement des outils qui lui sont fournis.

4) Le participant doit programmer la séquence du *réveil matinal* dans DOMUS à l'aide des outils qu'il vient de prendre en main (voir encadré de la figure 10). Le scénario est inscrit sur papier et lu par l'expérimentateur. Nous avons pris soin de limiter les mots similaires entre la séquence et l'éditeur de phrases. La séquence comporte également des actions irréalisables dans le cadre de DOMUS et peut nécessiter des modifications dans les phrases déjà rédigées. De façon à rassurer le participant, l'expérimentateur reste présent durant toute la programmation, mais ne répond plus aux questions posées par le participant. Il l'encourage cependant à formuler à haute voix ses actions et interrogations de façon à en prendre note. Lorsque le participant a suffisamment confiance dans le programme qu'il a développé, il peut aller le tester dans DOMUS accompagné de l'expérimentateur.

5) L'expérimentateur vérifie avec le participant le bon déroulement du scénario. Si ce n'est pas le cas, il le questionne sur la raison de l'écart entre le comportement constaté de l'appartement et celui qui est attendu. Le participant apporte à son programme les modifications souhaitées et teste de nouveau.

6) Le participant est convié à un débriefing au cours duquel il peut faire part de ses impressions sur l'exercice qu'il vient d'effectuer et sur KISS. C'est également l'occasion pour l'expérimentateur de revenir sur les actions et interrogations du participant notées à l'étape 4. Enfin, avant de libérer le participant, il lui est demandé de remplir un questionnaire d'évaluation (joint en Annexe 4).

Ces six étapes durent au maximum 2 heures et nécessitent la présence de deux expérimentateurs : l'un en tant que compère, l'autre pour accompagner et guider le participant durant le test.

5.2 Recrutement des participants

Nous avons effectué notre test d'acceptabilité avec 13 participants recrutés par courriel et par relation personnelle. Notre échantillon comprend six femmes et sept hommes. Comme nous souhaitions avoir de « véritables » utilisateurs finaux, nous avons limité à 4 le nombre de professionnels informaticiens. Parmi les neuf autres, cinq sont des cadres supérieurs, et quatre ont une profession intermédiaire. Tous nos participants ont une source de revenu. La moyenne d'âge de notre échantillon est de 39 ans. La tranche d'âge est comprise entre 26 et 66 ans en suivant la répartition donnée dans le tableau 1.

Tableau 1. Répartition des âges dans l'échantillon de participants.

Sujets	Tranche d'âges	Professions et Catégories Sociales
S1	26 à 30 ans	Professions Intermédiaires
S2	40 à 66 ans	Cadres supérieurs
S3	26 à 30 ans	Professions Intermédiaires - Informaticien
S4	26 à 30 ans	Professions Intermédiaires - Informaticien
S5	40 à 66 ans	Cadres supérieurs - Retraité
S6	30 à 40 ans	Professions Intermédiaires
S7	30 à 40 ans	Professions Intermédiaires - Informaticien
S8	40 à 66 ans	Cadres supérieurs - Retraité
S9	26 à 30 ans	Professions Intermédiaires - Informaticien
S10	30 à 40 ans	Professions Intermédiaires
S11	40 à 66 ans	Professions Intermédiaires
S12	30 à 40 ans	Employés-Ouvriers
S13	30 à 40 ans	Professions Intermédiaires

Le corpus de données comprend 26 heures d'enregistrement vidéo et audio accompagnées d'une grille d'analyse des questionnaires et d'une analyse thématique.

6 Résultats du test d'acceptabilité

Nous présentons ici les résultats du test d'acceptabilité en trois parties. La première porte sur les aspects positifs retenus par les participants, la seconde décrit les difficultés rencontrées suivie d'une discussion sur les pistes d'amélioration

6.1 Résultats : aspects positifs

Expérience et contrôle appréciés

Les participants ont apprécié l'expérience. L'interaction avec l'habitat est jugée « *amusante* » (S3, S6), « *sympathique* » (S9, S12), « *ludique* » (S13). Ils ont clairement manifesté l'envie d'acquérir un tel système, mais pas sous n'importe quelle condition. Trois participants ont précisé qu'un « *système facilitant c'est bien, mais de temps à autre, il peut être agréable de pouvoir reprendre la main* » (S5, S9), « *Ça, c'est fait pour faciliter la vie, mais j'aime bien maîtriser ce que je fais. Le dimanche, j'ouvre les volets moi-même* » (S10).

Cette appréciation générale confirme les résultats de notre étude amont [Coutaz 10] et confirme la nécessité de disposer de points de contrôles ouverts sur l'utilisateur tels que définis dans le projet CONTINUUM avec le concept de méta-IHM.

Bonne accessibilité du LPN

Pour l'ensemble des participants (13/13), le LPN est cohérent, « *compréhensible, pas ambigu* » (S3, S4). L'éditeur de phrases a été jugé « *assez précis, facilement manipulable* » (S9), et « *relativement intuitif* » (S11). Seuls deux sujets l'ont trouvé inutilement complexe (S5 et S8). Cependant, pour S8, le résultat du questionnaire doit être mis en regard de ses commentaires : « *tout a été facile, ça se compliquerait s'il y avait d'autres options* » (S8). En d'autres termes, il semble que S8, bien que non informaticien, anticipe la difficulté d'usage du LPN et de sa technique d'interaction par menus avec le passage à l'échelle.

Les participants se sentent capables d'utiliser le LPN sans l'assistance d'une personne technique (11/13). Ils pensent également que la plupart des utilisateurs peut apprendre à l'utiliser rapidement (11/13) : « *c'était accessible pour moi* » (S10), « *facile à comprendre* » (S6), « *compréhensible car c'est du langage parlé, c'est très vivant* » (S2). Ils indiquent ne pas avoir eu besoin de beaucoup de choses avant de pouvoir l'utiliser (9/13) : « *Ça rapproche le raisonnement informatique de notre raisonnement à nous* » (S2), « *Ça exprime ce qu'on va faire naturellement, comme on se l'imagine* » (S9).

Bonne compréhension des rôles du LPN et de l'habitat virtuel

À l'exception de trois personnes, les participants (10/13) affirment que l'utilisation conjointe de l'éditeur de phrases et de l'habitat virtuel est adaptée à la programmation d'un habitat. Parmi les trois opposants (S1, S7, S13), S7 et S13 se contenteraient du LPN. Sept autres pensent que ce langage est suffisant mais que l'habitat virtuel est un bon complément pour programmer. Cela porte à 9 (sur 13) le nombre de participants d'accord pour affirmer que le LPN est suffisant pour programmer l'appartement. Seuls 4 participants ont besoin de l'habitat virtuel (S2, S5, S11, S12). Nous avons donc 3 catégories d'utilisateurs :

1. Ceux qui ont besoin du langage pseudonaturel et de l'habitat virtuel,
2. Ceux qui n'ont besoin que du langage,
3. Ceux qui ont besoin du langage, mais pour lesquels l'habitat virtuel est un bon complément.

Il est possible qu'un effet d'âge explique en partie l'existence de ces trois catégories. En effet, la moyenne d'âge de la catégorie *a* est de 47 ans alors que celles des deux autres catégories sont respectivement 34 et 35 ans.

Ces résultats sont cohérents avec les fréquences d'utilisation des outils : quatre participants aimeraient utiliser fréquemment l'habitat virtuel (S2, S4, S5, S12) parmi lesquels nous retrouvons trois des quatre participants (S2, S5, S12) à avoir exprimé le besoin d'utiliser le LPN ainsi que l'habitat virtuel pour programmer DOMUS. Le nombre de participants indiquant vouloir utiliser fréquemment le LPN est, quant à lui, plus élevé (9/13). Ce positionnement paraît logique puisque seul ce langage a un réel impact sur l'appartement intelligent. Cependant, quatre participants (S2, S7, S8, S11) disent ne pas souhaiter utiliser le LPN fréquemment. Pour trois d'entre eux, la raison en est qu'ils veulent le faire « *une fois pour toute* » ou y revenir « *rarement* » (S8). Notons que deux autres participants ont exprimé oralement une remarque similaire bien que leurs réponses au questionnaire ne le montrent pas

L'utilité de l'habitat virtuel a été comprise de tous : « *ça va me servir à vérifier que ça marche sans me déplacer* » (S1), « *j'aime bien rester assis sur ma chaise* » (S9), « *pas besoin de se lever* » (S8), « *on peut voir, revoir, et rejouer le scénario jusqu'à ce qu'il corresponde à ce que l'on veut programmer* » (S11), « *le virtuel montre bien que le réel va fonctionne*r » (S12), « *en virtuel, la machine à café, ça fonctionne. Pas besoin de voir en vrai* » (S9). Toutefois notons que S8, non informaticien, estime que « *dans un système fiable, cet outil* [l'habitat virtuel] *ne sert à rien* » !

Les participants, unanimement, attestent que l'habitat virtuel permet d'identifier correctement les objets (13/13). Il en va de même pour le vocabulaire utilisé : *« À chaque fois, j'ai pu trouver ce que je voulais, même si ce n'était pas les mêmes mots que dans le scénario* » (S10)..

Habitat virtuel 2D et habitat virtuel 3D : préférence pour le 2D en raison de son efficacité

De manière informelle, nous avons demandé aux participants leurs impressions sur l'habitat virtuel 3D que nous avions prévu à l'origine. Celui-ci a été jugé plus réaliste que l'habitat 2D mais « *lourd* » (S9), « *trop compliqué* » (S7), « *moins pratique que le plan 2D* » (S12). « *Le 3D est un jeu ; là, on veut être efficace ; le plan grossier 2D est génial* » (S2). Un sujet seulement semble préférer la 3D : *« les gens commencent à être habitués au 3D, avec les visites de musée* »(S5). De manière générale, les participants partagent l'avis que le plan 2D est mieux adapté que l'interface 3D pour tester un programme rapidement et efficacement.

6.2 Difficultés rencontrées

Raisonnement *non sequitur*

Deux participants (S1 et S9) menaient des raisonnements *non sequitur* sur certaines conditions. S9, qui est informaticien, explique ceci : « *Quand je rentre, il se passe une action, et quand je sors, l'action est toujours en cours. Alors que quand je suis dans implique : tant que je suis dans il se passe une action et quand je n'y suis plus l'action s'arrête* ». Par exemple, « *Quand je suis dans la salle de bain, allumer la lumière de la salle de bain* » implique « *Quand je ne suis pas dans la salle de bain, éteindre la lumière de la salle de bain* ». Ce type de raisonnement est aussi apparu lors de la

spécification de règles portant sur l'horaire. En particulier, la phrase « *Quand il est entre 7h et 8h, allumer la lumière* » a été interprétée par S1 comme « *Tant qu'il est entre 7h et 8h, allumer la lumière, l'éteindre sinon* ».

Une explication possible à ces raisonnements réside probablement dans la différence d'aspect des verbes. En effet, le verbe de la condition « *quand je rentre* » est d'aspect inchoatif alors les verbes des expressions « *je suis dans* » et « *il est entre 7h et 8h* » sont d'aspect duratif. Les participants ont toujours interprété correctement le cas inchoatif « *quand je rentre* » alors qu'ils ont parfois mal interprété les cas duratifs « *je suis dans* » et « *il est entre* ». Il est donc possible que l'aspect (inchoatif/duratif) de la condition ait un impact sur l'interprétation qui en est faite.

Non conformité du vocabulaire

Trois participants ont trouvé le vocabulaire du LPN limité (S5, S11, S13). Par exemple, il n'est pas possible d'appliquer une action à un ensemble d'objets comme dans « *éteindre toutes les lampes* » (S11), ou inversement de désigner un équipement spécifique parmi plusieurs : « *les volets, les volets d'où ?* » (S5, rejoint par S13). Enfin, pour (S5), le LPN «*manque de personnalisation, trop mécanique, manque d'humanité*».

Une difficulté ressentie par les participants est de « *s'approprier un vocabulaire* » (S4) nouveau ou dont la sémantique est différente de la leur. Par exemple, l'action *allumer* est, dans KISS, l'unique moyen fourni à l'utilisateur pour modifier l'intensité des lumières. Lorsque les participants ont voulu programmer l'augmentation de l'intensité lumineuse, deux problèmes se sont posé à eux : trouver le verbe d'action (S6, S7, S10) car « *ce n'est pas l'action « allumer » puisque les lumières sont déjà allumées. Je veux juste augmenter* » (S7) ; et comprendre les différents niveaux d'intensité : « *ce n'est pas ma notion d'intensité* » (S13).

Ces manques de conformité du LPN au regard du vocabulaire de la langue naturelle peuvent expliquer le manque de confiance en soi : « DOMUS *a compris ce que je lui ai dit, c'est moi qui me suis trompé* » (S13). « *L'appartement a compris, c'est moi qui ai eu du mal à dire ce que je voulais* » (S11), « *L'appartement a compris ce que je lui ai dit, c'est moi qui me suis trompé* » (S12), « *L'appartement a compris, j'ai fait des erreurs* » (S13), « *La maison a réagi à ce que j'ai dit, c'est ma faute, je me suis trompé* » (S6). Seul un sujet a exprimé le problème dans l'autre sens : « *C'est nous qui nous nous adaptons au système, malheureusement* » (S9).

Les participants se sentent toutefois capables de surmonter ces difficultés avec le temps. « *Le premier coup est un peu galère, le second est génial* » (S5), « *après deux trois fois, ça va* »(S8), « *quand on est habitué, ça va* » (S12), « *à force de faire, j'y arriverai mieux* » (S13). Sept participants ont exprimé cet « effet apprentissage ».

Non conformité du modèle d'exécution orienté règles et séquencement des phrases

Si la traduction de chaque phrase du scénario en une phrase LPN est perçue comme facile, une suite de phrases, qui se lit à la suite en langue naturelle, peut se comprendre comme une suite d'instructions à exécuter en séquence, non pas comme un ensemble de règles dont l'ordre d'exécution est déterminé par la véracité des conditions.

Ainsi, certains participants ont pensé que l'ordre des phrases avait une importance. Par exemple, (S10) a souhaité déplacer une phrase entre deux autres. Pour (S4) et (S7), certaines phrases ont du sens, si et seulement si d'autres phrases ont été exécutées préalablement. (S2) explique pourquoi il n'a rédigé qu'une phrase avec un horaire : « *j'ai supposé que tout était lié, j'ai commencé avec un*

horaire, tout en découle » (S2). D'autres auraient apprécié définir des contraintes d'ordre : « *j'avais envie de dire après* » (S11, S3), « *d'ajouter un délai de 15 minutes* » (S5, S7), « *de dire si je suis encore dans mon lit après que la règle 1 a été enclenchée* » (S7).

Complexité implicite de l'activité de programmation

Plusieurs participants ont exprimé leur peur « *d'oublier quelque chose* » (S3, S10, S11, S12). Pour l'un des participants, cela justifie la non-utilisation du LPN : « *car je pense qu'on oublierait toujours quelque chose* » (S11). Un autre participant pointe la nécessité d'être bien concentré pour ne rien oublier : *« il ne faut pas le faire à 11h du soir* » (S10). Cela paraît toutefois surmontable (S13) : « *prendre en compte tous les paramètres, c'est une gymnastique de l'esprit, mais ce n'est pas vraiment dur* ».

Bien que chaque participant n'ait rédigé que sept phrases au plus, sept des treize participants ont rédigé deux fois la même phrase en pensant en rédiger deux différentes. En situation réelle, le problème risque d'être d'autant plus prégnant que le nombre de phrases à rédiger sera élevé : « *Dans une grosse maison, ça deviendrait compliqué* » (S5).

Besoin de services à valeur ajoutée

Certains participants ont émis des réserves quant à l'intérêt d'un habitat intelligent ou de rédiger un programme pour « *allumer trois lumières* » (S5). « *Si c'est pour allumer les lumières, j'appuie déjà sur un interrupteur, allumer les lumières automatiquement ne me fait pas gagner du temps. Par contre, pour quelque chose qui me fait gagner du temps sur des tâches que je ne veux pas faire : repasser, cuisiner, là ça a un intérêt* » (S1).

Les utilisateurs sont donc prêts à s'investir dans des activités de programmation s'il y a une valeur ajoutée : encore un résultat conforme à notre étude amont [Coutaz 10] et à la théorie de Blackwell su rl'investissement attentionnel [Blackwell 02].

Absence de support multi-habitants

Enfin, quelques sujets se sont inquiétés de l'usage de KISS dans un domicile habité par une famille. Les questionnements portaient sur le lien entre les programmes des différentes personnes (S8) et en particulier comment les conflits entre des règles contradictoires pouvaient être résolus (S5). Ces inquiétudes touchent l'une des limitations de ce travail, à savoir que nous nous sommes restreints à l'étude de la programmation de l'habitat par un habitant unique.

6.3 Pistes d'améliorations

Personnalisation et extension du LPN

Il ressort de l'expérimentation que le vocabulaire de l'éditeur de phrases doit être adaptable de deux manières complémentaires : renommage des éléments du vocabulaire et extensions du langage.

L'extensibilité de KISS est assurée via la Base de Connaissances (BdC) qui informe dynamiquement KISS sur la nature des entités de l'espace intelligent manipulables par l'utilisateur. Le renommage des entités et l'ajout d'expressions comme « tous les [volets]», « sauf les [volets] », ou les qualificatifs comme dans « volets exposés sud » nécessitent l'enrichissement de la BdC et l'adaptation de la grammaire. KISS inclut un éditeur de grammaire, mais en l'état, son IHM n'est pas destinée à un utilisateur final (voir Annexe 2). De plus, si la grammaire est structurée de façon à distinguer les

unités syntaxiques génériques de celles qui dépendent du domaine, l'implémentation actuelle a recours à un fichier XML écrit à la main alors qu'il pourrait être généré à partir de la BdC.

L'enrichissement de la BdC et l'adaptation de la grammaire sont envisageables par apprentissage où l'utilisateur et le système apprennent l'un de l'autre comme dans TellMe [Gil 11]. TellMe essaie d'associer les termes qu'il ne comprend pas à des concepts qu'il connaît en questionnant périodiquement l'utilisateur, ceci au risque d'être disruptif.

Base de programmes réutilisables et modifiables

Regroupement de phrases. Plusieurs participants ont exprimé le besoin de regrouper des phrases, soit en fonction de leurs dépendances (S4, S7, S11) : « *grouper les étapes entre elles* » (S7), soit en fonction de conditions similaires (S2, S3), soit par fonctionnalités, par exemple les phrases liées à l'économie d'énergie (S5). Certains ont émis l'idée que des groupes de phrases pourraient avoir priorité sur d'autres.

Regroupements de phrases patrons. Il a aussi été proposé que l'habitat soit « livré » avec un ensemble de phrases « *par défaut* » (S7) encodant « *des choses basiques* » pour définir un comportement de confort « *minimum* », mais paramétrables ou modifiables pour les exceptions. Par exemple, pour (S5), l'utilisateur va programmer deux scénarios correspondant à sa routine matinale : le scénario de base, avec seulement la gestion des lumières par exemple, et le scénario tout confort, qui comprend le scénario de base plus toutes les briques nécessaires au confort. « *Le scénario tout confort s'exécutera dans 80% des cas, quand tout se passera bien, quand l'utilisateur sera à l'heure, et qu'il n'y aura pas de tempête dehors, etc. Le scénario de base sera, lui, utilisé dans les 20% de cas restants* ». Cet ensemble de phrases serait modifiable et extensible : « *allumer la lumière de la pièce quand j'y suis, c'est de base, après je mets une heure* » (S2). Plusieurs participants ont généralisé cela en suggérant l'utilisation de phrases « patrons », paramétrables (S1, S2, S5, S8).

Ces regroupements confirment le besoin de « paquets de services paramétrables » observés lors de notre étude amont de terrain avec prise en compte des exceptions [Coutaz 10].

Détection des contradictions sémantiques

L'éditeur LPN permet de rédiger des phrases paradoxales comme : « *allumer la lumière de la chambre et éteindre la lumière de la chambre* » (S4), « *quand il est 7h et quand il est 8h* » (S1), « *quand je suis dans la salle de bains et quand je suis dans la cuisine* » (S2).

D'autres formes de guidage ont été suggérées par les participants comme le fait que le système puisse faire des rappels du type « *le lieu de l'action n'est pas spécifié* » (S5), « *vous n'avez pas précisé d'heure* » (S10). Il s'agit-là d'une réelle faiblesse fonctionnelle de KISS.

Généralisation du multisyntaxe avec égale opportunité

Notre hypothèse d'un éditeur multisyntaxe se confirme, mais à condition d'assurer une totale égale opportunité entre les syntaxes. L'architecture de KISS, qui s'appuie sur un langage pivot, permet cette ouverture. L'intérêt du multisyntaxe est de laisser le choix à l'utilisateur et ceci à tout instant. Sur le plan interactionnel, le multisyntaxe implique que l'utilisateur a la possibilité de spécifier une instruction de manière multimodale, soit en équivalence, complémentarité ou redondance (cf. les propriétés CARE [Coutaz 95]).

Une première amélioration porte sur la modalité menu. Pour (S3), «*la manipulation via les menus est fastidieuse* ». À l'évidence, les menus actuels de KISS conviennent au débutant pour ses propriétés de feedforward et de support à la correction, mais ils ne sont pas adaptés à l'utilisateur expert. Cette

faiblesse est facile à corriger : ajout d'un mode expert (à-la-marking menu [Kurtenbach 93]) augmenté de listes déroulantes à-la iPhone pour le passage à l'échelle.

Une seconde amélioration est l'introduction de nouvelles modalités : (S2) a souhaité que la programmation se fasse vocalement : « *j'aurais préféré du parler et de la magie* » ou bien la saisie de texte semi-libre avec suggestions et terminaisons automatiques.

Une troisième amélioration est l'utilisation de la complémentarité entre texte (oral ou écrit) et langage visuel iconique ou formulaire, ou encore entre texte et spécification sur exemple et par l'exemple [Girard 92] au moyen de l'habitat virtuel :

- Pour la pose de contraintes horaires, (S12) a suggéré de modifier le texte « *il est* », présent dans le premier menu des conditions, par une icône représentant une horloge. Un autre participant a proposé un tableur avec des horaires « *à la manière d'un doodle* ».
- (S4) aurait souhaité l'usage de cases à cocher pour spécifier de manière efficace des lieux d'exécution d'actions comme dans « *ouvrir les volets de la chambre et de la cuisine, mais pas ceux de la salle de bain* ». Concernant la spécification des localisations, plusieurs participants ont cherché à manipuler l'avatar Homer de l'habitat virtuel, espérant ainsi contraindre la localisation des actions mentionnées dans la phrase en cours de rédaction. Par exemple, déplacer l'avatar dans la chambre pour signifier qu'il faut « *allumer les lumières de la chambre* » ou encore « *placer Homer dans le lit pour contraindre la localisation des phrases au lit* » (S5).

Par extension, les utilisateurs voudraient que le système fasse « *la démonstration de ce que l'on a fait* » (S12). Ils seraient tentés de manipuler l'habitat virtuel pour programmer (S1, S2, S5, S7, S11) : « *avec des menus textuels sur les objets du monde virtuel* » (S1) où l'utilisateur « *joue dans le virtuel et clique-droit pour savoir quoi faire* » (S7). « *Programmer grossièrement avec le virtuel, puis au besoin, affiner avec les phrases* » (S5) : on retrouve le besoin de complémentarité entre les modalités de spécification des programmes

7 Conclusion

L'évaluation expérimentale du scénario prospectif de CONTINUUM appelle deux remarques essentielles. L'une concerne le respect de conditions expérimentales extrêmement exigeantes, l'autre porte sur le concept d'outil de développement par l'utilisateur final (DUF).

À propos des conditions expérimentales

Le déploiement d'un système ambiant, tel CONTINUUM, dans un environnement inconnu des développeurs de ce système (en l'occurrence, DOMUS), requiert de nombreuses adaptations. Il convient de jouer sur les compromis de façon à :

- assurer une robustesse logicielle et matérielle suffisante,
- satisfaire les requis de latence,
- respecter le réalisme et la pertinence écologique du recueil des données.

Dans notre cas, nous avons étudié le nécessaire équilibre entre de nouveaux développements logiciels ad-hoc (comme notre middleware de secours) et la révision du scénario expérimental.

Nous retenons de cette expérience deux leçons essentielles : (1) Prévoir dans la mise en œuvre d'un prototype de système ambiant l'intégration d'un composant Magicien d'Oz de façon à remplacer les entités défaillantes (ou non implémentées) par un compère humain. (2) Imposer la présence sur le terrain expérimental de toute l'équipe de développement du logiciel à tester de façon à corriger rapidement les bugs, pouvant éviter des développements ad-hoc inutiles.

À propos du DUF KISS

L'expérimentation démontre que les participants ont pu programmer avec succès le scénario imposé. L'utilisation du langage pseudo-naturel et l'utilité d'un habitat virtuel dual de l'habitat réel ont été comprises et validées.

Les difficultés rencontrées militent pour deux améliorations essentielles : (1) pousser la personnalisation et l'extensibilité du langage par co-construction système-utilisateur (approche par apprentissage) ; (2) pousser comme nous le pensions, l'édition multisyntaxe avec égale opportunité totale et modalités de programmation complémentaires. En particulier, l'habitat virtuel présenté aujourd'hui comme metteur au point peut aussi servir d'outil de spécification par démonstration en différé ou en temps réel. En mode « temps réel », l'habitat virtuel devient une « télécommande » avec effet immédiat sur le monde réel. Il s'agit alors d'une méta-IHM réflexe.

Au final l'habitat virtuel peut jouer trois rôles : metteur au point, support à la programmation par démonstration (méta-IHM tactique), télécommande de l'habitat réel (ou méta-IHM réflexe). Ce glissement de rôle rendu possible par l'architecture de CONTINUUM et de KISS, est un résultat original par rapport à l'état de l'art.

8 Références bibliographiques

[Ash 11] Ash, J., Babes, M. Cohen, G., Jalal, S. et al. Scratchable Devices; user-Friendly Programming for Household Apliances. In *Proc. Human-Computer Interaction, Part III, HCII 2011*, J.A. Jacko (Ed.), LNCS 6763, Springer-Verlag Berlin Heidelberg Pub., p. 137–146, 2011.

[Blackwell 04] Blackwell, A.F. End-user developers at Home. *Communication of the ACM*, sept. 2004, Vol.47, no9, 2004, 65-66.

[Blackwell 01] Blackwell, A.F. & Hague, R. AutoHAN: An architecture for programming the home. In Proc. of the *IEEE 2001 Symposium on Human-Centric Computing Languages and Environments*, IEEE Pub., 2001.

[Blackwell 02] Blackwell, A. & Burnett, M. Applying attention investment to end-user programming. In *Proc. of the IEEE 2002 Symposium on Human Centric Computing Languages and Environment*s, p.28-30, 2002.

[Brandt 08] Brandt, J., Guo, P., Lewenstein, J. & Klemmer, S. Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice. In Proc. of the *4th workshop on end-user software engineering (WEUSE 08)*. ACM Pub., 2008, p. 1-5.

[Calvary 07] Calvary, G. *Plasticité des Interfaces Homme-Machine*. Thèse HDR de l'Université de Grenoble, 2007.

[Cockton 04] Cockton, G. 2004. From quality in use to value in the world. *Extended abstracts of the 2004 conference on Human factors and computing systems - CHI '04*, 2004.

[Cockton 05] Cockton, G. 2005. A development framework for value-centred design. *In CHI '05 extended abstracts on Human factors in computing systems - CHI '05*. New York, New York, USA: ACM Press, 2005.

[Coutaz 95] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. & Young, R.M. Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. *Proceedings of INTERACT*, vol. 95, no. June, pages 115–120, 1995.

[Coutaz 05] Coutaz, J., Crowley, J.L., Dobson, S. & Garlan, D. Context is key. *Communications of the ACM*, vol. 48, no. 3, p. 49–53, 2005

[Coutaz 06] Coutaz, J. Meta-UI for Ambient Spaces. In *Proc. TAMODIA 2006*, LNCS 4385, p. 1-15, Springer-Verlag Heidelberg 2007.

[Coutaz 10] Coutaz, J., Fontaine, E., Mandran, N., Demeure, A. DisQo: a user needs analysis method for smart home. In *Proc. Nordi CHI2010*, ACM Pub., 2010.

[Dahlbäck 92] Dahlbäck, N., Jönsson, A & Ahrenberg, L. Wizard of Oz studies — why and how. *Third Conference on Applied Natural Language Processing*, Trento, Italy, 31 march—3 April, 1992.

[Davidoff 06] Davidoff, S., Lee, M.K., Yiu, C. Zimmerman, J. & Dey, A. Principles of Smart Home Control. *In Ubicomp 2006, LNCS 4206, Springer Verlag Berlin Heidelberg*, 19-34, 2006.

[Demeure 08] Demeure, A., Calvary, G. & Coninx, K. COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces. In *Proc. 15th International Workshop, DSV-IS 2008*, T.C.N. Graham & P. Palanque (Eds), Lecture Notes in Computer Science 5136, Springer Berlin / Heidelberg, Kingston, Canada, July 16-18, p. 225-237, 2008.

[Dey 06] Dey, A., Sohn, T., Streng, S. & Kodama, J. 2006. iCAP: Interactive prototyping of context-aware applications. In *Proc. Pervasive Computing*, Springer Verlag, 254–271.

[Drey 10] Drey, Z. *Vers une méthodologie dédié à l'orchestration d'entitées communicantes*. Thèse Université de Bordeaux, 2006.

[Ferry 10] Ferry, N., Lavirotte, S., Tigli, J.-Y., Rey, G. & Riveill, M. Toward a Behavioral Decomposition for Context-awareness and Continuity of Services. In *Proc. Ambient Intelligence and Future Trends -*

International Symposium on Ambient Intelligence (ISAmI), *Advances in Intelligent and Software Computing*, Springer. 978-3-642-13267-4, p. 55-62, 2010.

[Harrison 96] Harrison, S. & Dourish, P. Re-place-ing space: the roles of place and space in collaborative systems. In *Proc. of the 1996 ACM conference on Computer Supported Cooperative Work*, CSCW 96, p. 67-76, ACM pub., 1996.

[Gabillon 11] Gabillon, Y. *Composition d'Interfaces Homme-Machine par planification automatique*. Thèse de l'Université de Grenoble, 2011.

[Gil 11] Gil, Y., Ratnakar, V. and Frtiz, C. TellMe: learning procedures from tutorial instruction. In *Proc. of the 16th international conference on Intelligent user interfaces* (IUI'11), 227-236, 2011.

[Girard 92] Girard, P. *Environnement de programmation pour non programmeurs et paramétrage en conception assistée par ordinateur : Le système Like*. Thèse HDR, Nov. 1992. Univ. Poitiers.

[Kierkegaard 11] Kierkegaard, P. Beefing Up End User Development: Legal Protection and Regulatory Compliance. In *Proc. third international symposium on End-User Development*, IS-EUD 2011, Springer Verlag Pub., p. 203-217, 2011.

[Ko 11] Ko, A.J. et al. The State of the Art in End-User Software Engineering. *ACM Computing Surveys*, Vol. 43, No 3, article 21, ACM pub., 2011.

[Kurtenback 93] Kurtenbach, G., Sellen, A. & Buxton, W. An Empirical Evaluation of Some Articulatory and Cognitive Aspects of Marking Menus. *Human-Computer Interaction*, vol. 8, no. 1, p. 1–23, 1993.

[Lieberman 06] Lieberman, H., Paternò, F. and Wulf, V. End-User Development. *Human-Computer Interactions Series*, Vol. 9, Springer, 2006.

[Nardi 95] Nardi, B., A. *A Small Matter of Programming – Perspectives on end user computing*. The MIT Press, Cmabridge, Massachussets, 1995 (second edition).

[Preece 02] Preece, J., Rogers, Y. & Sharp, H. *Interaction Design: Beyond Human-Computer Interaction*. Wiley & Sons pub., 2002.

[Repenning 11] Repenning, A., Ahmadi, N. & Repenning, N. Collective Programming: Making End-User Programming (more) Social. In *Proc. third international symposium on End-User Development*, IS-EUD 2011, Springer Verlag Pub., 2011.

[Resnik 09] Resnik, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch : Programming for all. *Communications of the ACM*, Nov. 2009, Vol. 52, No11, 2009, 60-67.

[Rey 05] Rey, G. *Contexte en Interaction Homme-Machine : le contexteur*. Thèse de doctorat de l'Université de Grenoble, 2005.

[Sohn 06] Sohn, T.Y., Dey, A.K., iCAP: An Informal Tool for Interactive Prototyping of Context-Aware Applications. In *Proc. of the International Conference on Pervasive Computing 2006*. Dublin, Ireland, May 2006, 974–975.

[Sousa 11] Sousa, J.P. et al. End-User development of software systems for smart spaces. In *International Journal of Space-Based and Situated Computing (IJSBSC)*, Inderscience, 2011.

[Tennant 83] Tennant, H.R., Ross, K. and Thompson, C. Usable natural language interfaces through menu-based natural language understanding. In *Proc. Computer Human Interaction*, CHI'83, ACM Publ., 1983, 154-160.

[Truong 04] Truong, K.N., Huang, E.M, and Abowd, G. CAMP : a magnetic poetry interface for end-user programming of capture applications for the home. In *Proc. of the 6th international conf. on ubiquitous computing*, Ubicomp'04, 2004, 143-160.

9 Annexe 1 – Scénario prospectif

mGenius est la forme moderne du Génie de la mythologie romaine, divinité accompagnant chaque individu tout au long de sa vie, le guidant et veillant sur lui à chaque instant. mGenius est un compagnon autonome d'aide à la vie de tous les jours, mais qui reste placé sous le contrôle de son propriétaire. Techniquement, mGenius est un ensemble dynamiquement reconfigurable de services logiciels centrés sur les besoins et attentes d'un utilisateur. Ces services exploitent de manière opportuniste les ressources de calcul, de communication et d'interaction disponibles. Il agit comme une aura computationnelle au service de son propriétaire quelle que soit la situation.

WeCo ou Ordinateur Porté/Wearable Computers : PDAs, Smartphones, etc., sont des déclinaisons technologiques de l'évolution du terminal utilisateur avec pour seule constante d'être constitué d'un ensemble de dispositifs personnels, aux multiples configurations possibles (téléphone mobile, oreillette, GPS, Head-mounted display, appareil photo externe, lecteur de carte à puce externe d'un porte monnaie électronique, etc.) à l'image d'un smartphone qui serait sécable. Cette notion n'est pas sans rappeler celle de « Wearable Computer » ou « ordinateur porté » définie dès les années 80 par Steve Mann. Nous choisissons donc d'appeler ce terminal utilisateur multi-dispositifs et multi-configurations : WeCo.

Acteur principal : Bob. Il habite Grenoble.
Acteurs secondaires : son amie Alice et des inconnus.
Situation : C'est l'été. Bob part en vacances à Vallauris où il rejoindra Alice pour un mois.

Moment 1 : à la maison

mGenius sait que c'est le jour de départ en vacances et qu'il est temps de se lever. Comme Bob est encore au lit et qu'il a demandé à être réveillé, mGenius active le système de réveil : il ne trouve pas le réveil (qui a dû être débranché) et choisit d'allumer le téléviseur situé dans la chambre en le réglant sur une chaîne musicale. La musique se met en route à faible volume, puis une lumière douce s'allume. Bob ne semblant pas se réveiller, mGenius, conformément aux préférences de Bob, emploie les grands moyens : l'intensité de chaque signal augmente. En parallèle, mGenius a déclenché la cafetière de sorte que le petit-déjeuner soit prêt.

Bob se réveille et commence à se préparer. Pendant son petit-déjeuner, mGenius informe Bob des conditions météo et de circulation (trafic, travaux) sur son trajet. Le téléphone sonne alors qu'il prend sa douche. Comme l'appel n'est pas urgent (interlocuteur non central dans la vie de Bob), mGenius décroche pour Bob et enregistre le message. Finissant ses préparatifs, Bob demande à mGenius de lui préparer une sélection de musiques, de films et de jeux qu'il utilisera pendant ses vacances. Les préparatifs terminés, Bob quitte son domicile.

mGenius verrouille la maison, redirige l'interphone sur le WeCo de Bob et active l'alarme. Le système de détection de présence bascule en mode absence (Bob part pour 1 mois). En cas d'intrusion, il avertira la société de gardiennage ainsi que Bob par tous les moyens possibles et le plus rapidement possible. En outre, mGenius règle l'arrosage automatique pour qu'il se déclenche en début de soirée si la pelouse est sèche et que personne ne se trouve sur le secteur arrosé (les enfants du voisin y jouent de temps en temps). En cas de panne du dispositif d'arrosage ou de coupure d'eau, mGenius contactera la société de gardiennage.

Moment 2 : entre la maison et la voiture à l'arrêt

Bob se dirige vers sa voiture. mGenius signale sur son WeCo qu'il a manqué un appel alors qu'il était dans la salle de bain. Bob demande à écouter le message. Tout en l'écoutant, il s'installe au volant. Au moment où Bob pose son WeCo, mGenius bascule la communication sur les haut-parleurs et active le micro pour que Bob puisse rappeler son interlocuteur. La conversation terminée, Bob démarre. mGenius met en route le GPS et rappelle les conditions météo sur le trajet au moyen des ressources de la voiture.

Moment 3 : en route

Lorsque les conditions de circulation sont difficiles, mGenius bascule les communications téléphoniques sur le répondeur. En chemin, Bob réalise qu'il a oublié les photos qu'il devait montrer à son amie Alice. Il demande à mGenius de rapatrier les photos « laissées » à la maison. Il n'a pas besoin de s'authentifier. Ceci est fait par l'usage-même de mGenius depuis la voiture.

Bob, sensible au respect de l'environnement, prend l'autoroute du Soleil classée « voie verte ». Il est automatiquement pris en charge par le service de régulation du trafic routier qui gère les conditions de circulation et de pollution (via des capteurs humidité, vitesse du vent, …). Des messages sont affichés sur les panneaux de signalisation (nouvelle vitesse à respecter, accidents, bouchons, pic de pollution, etc.). mGenius, conformément au choix de Bob, affiche ces mêmes informations sur le dispositif le plus approprié (tableau de bord, haut-parleurs, GPS, etc.) et surtout, rappelle les consignes de vitesse lorsque Bob ne les respecte pas.

Après Montélimar, Bob s'inquiète de la présence d'une fumée blanche dans le ciel sur sa droite. Il demande à mGenius de l'informer sur cette pollution atmosphérique. Il obtient très vite la réponse: il s'agit simplement de l'évaporation issue de la centrale de Tricastin.

Moment 4 : pause déjeuner

Vers midi, mGenius propose une liste de restaurants conformes aux préférences de Bob (gastronomie, prix, distance, confort, etc.). Bob, ayant décidé de bien profiter de ses vacances, choisit un restaurant gastronomique assez proche de l'autoroute. Après un agréable repas, il décide de régler la note avec son service de paiement bancaire automatique. Une authentification forte est requise pour utiliser le porte-monnaie électronique. Pour cela, Bob doit fournir une preuve supplémentaire de son identité. Compte tenu du lieu public, Bob ne peut pas utiliser la reconnaissance vocale. mGenius lui suggère d'utiliser le lecteur biométrique du WeCo ou de composer le mot de passe au moyen du clavier. Bob reprend la route.

Moment 5 : dans le hall de l'hôtel

Bob arrive à l'hôtel. Il pourrait se diriger directement vers sa chambre : dès le parking de l'hôtel, mGenius lui indique toutes les informations utiles pour gagner sa chambre et la clé numérique de la chambre est automatiquement transférée sur le WeCo de Bob. Toutefois, Bob préfère s'adresser à l'hôtesse car il aimerait être conseillé sur des activités de villégiature. Il y a la queue à la réception. Pour passer le temps, il consulte les informations touristiques publiées sur le grand mur du hall. Comme il est seul devant le mur, il peut naviguer dans l'espace d'informations et sauvegarde sur son WeCo quelques adresses intéressantes (curiosités, visites de musée, randonnées, loisirs sportifs, restaurants, etc.). Quelqu'un arrive et s'intéresse au mur : les informations murales sont maintenant disponibles sur le WeCo de sorte que Bob puisse continuer son exploration en privé. Arrive son tour. Son exploration des curiosités est suspendue.

Bob, proche du comptoir, est automatiquement identifié sur le PC de l'hôtesse qui lui confirme sa réservation. Comme Bob a permis à mGenius d'exporter les bonnes adresses qu'il vient de relever, l'hôtesse propose de lui réserver une activité de son choix. La réservation est confirmée par mGenius sur le WeCo de Bob.

Moment 6 : vers la chambre d'hôtel

Satisfait, Bob se dirige vers sa chambre. mGenius perçoit cette action et règle d'ores et déjà le confort de la chambre. Bob, à une dizaine de mètres de sa chambre, constate qu'une porte s'éclaire, lui facilitant ainsi le repérage de sa chambre. Cependant une autre personne le suit à quelques pas, et c'est en entendant son WeCo biper devant la porte qu'il a la confirmation qu'il s'agit bien de sa chambre. La porte s'ouvre automatiquement grâce à la clef électronique récupérée à l'arrivée. Bob entre.

Moment 7 : dans la chambre d'hôtel

mGenius a déjà réglé l'ambiance lumineuse et sonore de la chambre en respectant autant que possible les préférences de Bob. Il a notamment affiché la photo préférée qui rappelle le bon pays de Grenoble ! Les services disponibles sont automatiquement indiqués y compris l'exploration des informations touristiques commencée dans le hall de l'hôtel. Il réserve une table pour deux personnes au restaurant de l'hôtel.

Exténué par le voyage, il s'allonge et s'endort. Alice va bientôt arriver. mGenius le sait et réveille Bob avec les ressources disponibles… Il est l'heure d'accueillir Alice dans le hall de l'hôtel. Plus tard, ils iront tous les deux au restaurant.

Moment 8 : au restaurant

Il y a du monde. Entre deux plats, Bob propose de montrer à Alice ses dernières photos de montagne. Pour cela, le WeCo n'est pas très commode. mGenius rend les photos disponibles sur la table. Tous deux explorent les photos sur la table. Le service utilisé n'est pas tout à fait le même que celui de la maison.

Alice trouve la photo du glacier de Celliers particulière car elle lui rappelle celle de la Muzelle qu'elle a photographiée l'année dernière. Elle la récupère sur son WeCo et met à disposition celle de la Muzelle sur la table. Parce que Bob et Alice sont amis, ces échanges sont permis en toute transparence et sécurité…

En fin de repas, Bob demande à mGenius de lui fournir la note de restaurant qu'il valide. Comme il ne veut pas montrer l'addition à son amie, l'affichage a lieu sur le WeCo, non pas sur la table. Après ces délicieux instants passés avec son amie, Bob rejoint sa chambre …

Moment 9 : incident médical

Dans la nuit, lorsque Bob montre des signes alarmants de santé. Personne à qui se renseigner. Il fait appel à mGenius. Celui-ci lui signale que Grenoble est à 5h de route en voiture. Soit il rentre sur Grenoble, soit il consulte sur Vallauris. Les vacances étant planifiées depuis longtemps, il préfère rester sur Vallauris. mGenius lui propose plusieurs solutions : le médecin de garde ; les urgences à l'hôpital le plus proche, etc. Il opte pour le médecin de garde. mGenius le met en contact avec le médecin via son SmartPhome. Puis, Bob ayant obtenu un rendez-vous en urgence, un plan est affiché pour s'y rendre sur son WeCo. La pharmacie de garde est également indiquée et localisée. Alice est avertie pour qu'elle le conduise au bon endroit.

10 Annexe 2 – Mise en œuvre technique de KISS

10.1 Présentation générale

La figure A2.1 montre la décomposition fonctionnelle de KISS et ses relations de dépendance avec l'infrastructure d'exécution. KISS comprend :

- Une *Grammaire* de description du langage de programmation et les moyens d'éditer cette grammaire pour en ajuster manuellement les unités (noté *Editeur de grammaire* dans le schéma) avec son interface homme-machine, *EdGrIHM*.
- Un éditeur de programmes, noté *EdProg*, et ses deux IHM : *IHM.LPN* et *IHM.Iconic*. Ces IHM font appel à la base de *Programmes existants* pour en proposer la liste à l'utilisateur final qui, dès lors, peut les réutiliser et les éditer. À son tour, le noyau fonctionnel d'EdProg comprend un gestionnaire de la *représentation pivot* du langage de programmation (sorte de syntaxe abstraite) et deux *générateurs* qui assurent la traduction entre cette représentation et les deux syntaxes concrètes *LPN* et *Iconic*. Comme dans AutoHan [Blackwell 01], notre représentation pivot permet d'assurer la propriété multisyntaxe. Notons toutefois que dans la version actuelle de KISS, la traduction entre la représentation pivot et la syntaxe iconique n'est pas bidirectionnelle. En conséquence, l'égale opportunité entre les syntaxes est, pour l'instant, partielle.
- *AnalyseurLx-Sx* assure le rôle usuel d'analyseur lexical et syntaxique. Il utilise évidemment la grammaire, mais de manière plus originale, la Base de Connaissances (BdC) qui permet de répondre à la dynamicité de l'espace intelligent. *AnalyseurLx-Sx* est responsable de la construction de la représentation pivot.
- L'analyseur sémantique *AnalyseurSem* est sollicité par EdProg lorsque l'utilisateur valide la phrase (sur sélection du bouton *Ajouter phrase*). Il effectue une analyse sémantique complémentaire en s'appuyant sur la BdC avant de transmettre le résultat au générateur.
- Le générateur *Générateur*, en s'appuyant sur une grammaire de (des) langage(s) cible(s), assure le rôle usuel de traducteur entre la représentation abstraite d'un programme et une représentation compréhensible par la(les) machines cibles. Dans le cas de CONTINUUM, la machine cible est le Gestionnaire de Contexte (GC). Le résultat de cette traduction est, d'une part, rangé dans la base des programmes existants, et d'autre part transmise à l'infrastructure d'exécution.
- *Simulateur*, qui sert de metteur au point, utilise l'infrastructure cible comme support d'exécution. Il comprend un noyau fonctionnel qui reproduit le fonctionnement du monde réel, et une IHM qui traduit auprès de l'utilisateur final, l'état actuel de ce monde (IHM 2D de la figure 6 ou IHM 3D de la figure 7).

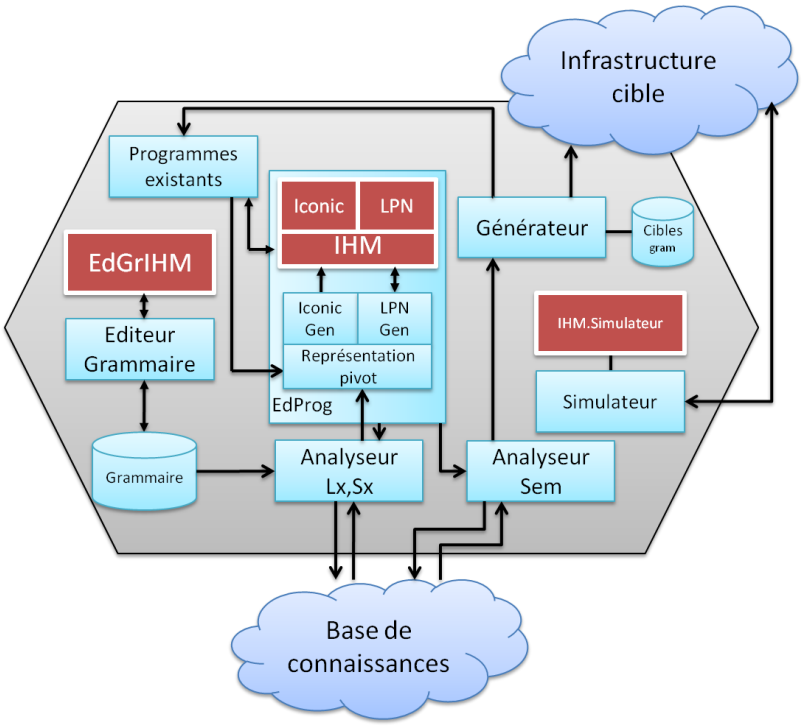


Figure A2.1. Décomposition fonctionnelle de KISS et ses relations de dépendance. Les boîtes dénotent les composants fonctionnels ; les cylindres, des informations rémanentes ; les flèches, les échanges d'informations et leur direction. La couleur de fond clair indique des noyaux fonctionnels tandis qu'une couleur de fond foncé désigne des IHM.

La figure A2.1 indique les deux points de dépendance de KISS avec son environnement d'exécution : la base de connaissances, dépendante du domaine d'application, et l'infrastructure qui assure l'adaptation dynamique de l'espace intelligent. La décomposition fonctionnelle de KISS montre comment nous avons minimisé ces dépendances via l'analyseur lexical et syntaxique, le générateur et le simulateur. Dans le cas de l'intégration de KISS dans CONTINUUM (voir figure A2-2) :

- L'analyseur Lx-Sx communique avec la BdC au moyen de requêtes SPARQL.
- Le générateur s'appuie sur la grammaire du langage cible, notée *Ctx-gram*, pour générer les plans d'adaptation sous forme de « situation-AAs ».
- Le simulateur utilise le protocole UPnP de telle sorte qu'un équipement (TV, radio, etc.) virtuel soit un dispositif UPnP. Ainsi, un équipement virtuel est représenté dans l'application par un composant proxy au même titre qu'un équipement réel. Ce faisant, nous obtenons une représentation numérique duale du monde réel.

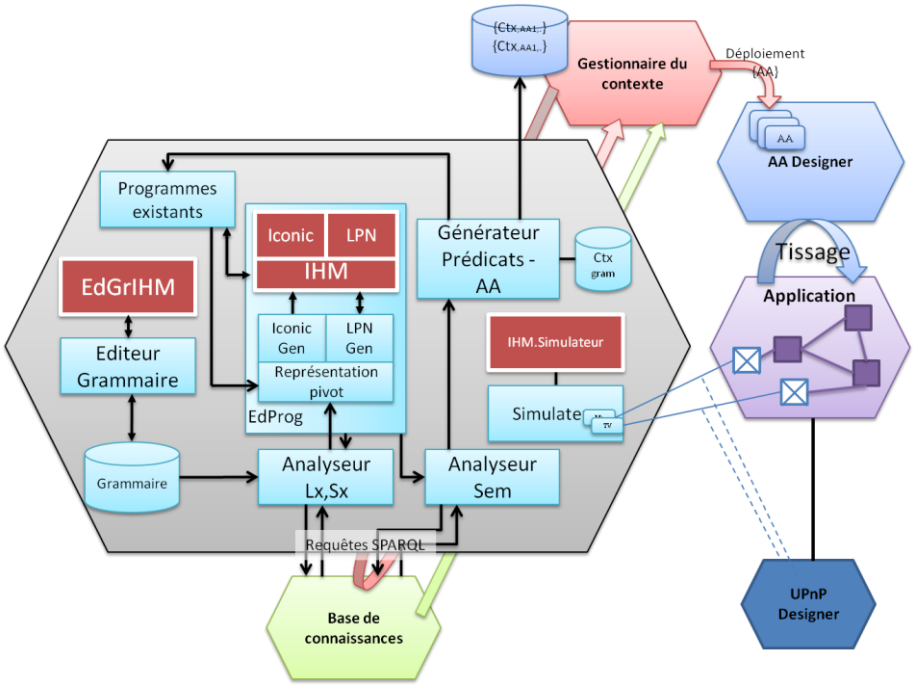


Figure A2-2. KISS installée dans CONTINUUM.

Nous décrivons maintenant en détail le fonctionnement des composants clefs de KISS en suivant les échanges d'information, depuis la grammaire du langage de programmation jusqu'au générateur d'AAs et au metteur point.

10.2 Grammaire KISS du langage de programmation

La grammaire KISS du langage de programmation est structurée en sous-grammaires de façon à séparer les éléments du langage qui dépendent du domaine de ceux qui n'en dépendent pas. De plus, les éléments dépendants du domaine recouvrent trois aspects qu'il convient de distinguer : les actions possibles dans le domaine (observer, ouvrir, fermer, etc.), les entités du domaine (équipements, personnes, etc.) et les éléments de contexte (localisation, date, etc.). Nous obtenons ainsi les quatre sous-grammaires suivantes :

- La *grammaire noyau*, indépendante de l'espace intelligent, sert de racine aux trois autres sous-grammaires.
- La *grammaire d'actions* décrit la structure grammaticale de chaque action possible dans l'espace intelligent, ceci indépendamment des instances d'entités sur lesquelles ces actions s'appliquent mais aussi indépendamment du contexte.
- La *grammaire des classes abstraites* traite de la désignation des entités de l'espace intelligent, ceci indépendamment du contexte.
- La *grammaire du contexte* qui exprime les situations contextuelles.

Les grammaires d'actions, des classes abstraites et du contexte sont enrichies dynamiquement à partir de la BdC. On assure ainsi la souplesse nécessaire à la prise en compte de la dynamicité de l'espace intelligent. En outre, le vocabulaire et la syntaxe sont modifiables au moyen de l'*Editeur de grammaire*. On assure ainsi la personnalisation et l'extensibilité lexicale, syntaxique et sémantique. Toutefois, dans sa version actuelle, EdGrammaire s'adresse à un utilisateur final averti car il nécessite de maîtriser le métalangage ABNF (Augmented Backus-Naur Form).

Nous avons étendu ABNF (Augmented Backus-Naur Form) de deux opérateurs :

- *Opérateur de redirection* pour désigner une référence à une source extérieure (une autre grammaire ou la BdC).
- *Opérateur sémantique* pour exprimer des traits sémantiques spécifiques au domaine et utilisés par l'analyseur sémantique.

Le tableau A2-1 récapitule l'ensemble des opérateurs de la grammaire KISS. La définition complète de la grammaire peut être consultée en fin de cette Annexe 2.

Tableau A2-2. Opérateurs utilisés dans la grammaire KISS.

Définition	=	ABNF
Alternative	/	ABNF
Non-terminal	Abz	ABNF
Terminal	" "	ABNF
Groupe	( )	ABNF
Optionnel	[ ]	ABNF
Sémantique	{ }	Extension
Redirection	! !	Extension
Répétition	*	ABNF non utilisé
Intervalle	-	ABNF non utilisé

La section suivante montre comment les éléments de langage de la grammaire KISS sont utilisés par l'analyseur lexical et syntaxique pour construire la représentation pivot et comment cette représentation pivot est utilisée par l'éditeur de programme.

10.3 Analyseur LX-SX et la représentation pivot

AnalyseurLx-Sx construit la représentation pivot qui modélise, sous forme d'arbre, les phrases permises du langage de programmation. Cette construction se fait en parcourant la grammaire KISS de manière incrémentale (pour suivre à la lettre les choix de l'utilisateur final via l'IHM.LPN et actualiser le feedforward) et fait appel à la BdC pour identifier les entités terminales spécifiques à l'espace intelligent.

Le point d'entrée du parcours de la grammaire est le non-terminal *sentence* de la sous-grammaire noyau. Comme le montre l'extrait du cadre A2-1, *sentence* comprend deux non-terminaux, *condition* et *action*, et porte une décoration sémantique, *{CONDITION:<1> ACTION:<2>}* qui, utilisée par l'analyseur sémantique en vue de la génération, sera détaillée plus loin. Nous allons expliquer le fonctionnement de l'analyseur Lx-Sx en détaillant la construction du sous-arbre correspondant à la partie *action* d'une phrase du langage de programmation.

La sous-grammaire noyau stipule qu'une *action* est une suite non vide d'*actionName* reliés par le terminal *et* et qu'un *actionName* est un non-terminal dont la définition se trouve dans la sous-grammaire *actionGram*. À son tour, la sous-grammaire d'actions définit un *actionName* comme une alternative de non-terminaux : *open*, *close*, *watch*, etc. Comme le montre la sous-grammaire d'actions, toutes les définitions de ces non-terminaux commencent par un terminal. L'analyseur Lx-Sx

en « sait » suffisamment pour fournir à EdProg la représentation pivot de la figure A2-3 à partir de laquelle EdProg génère le menu déroulant *ouvrir*, *fermer*, *regarder*, etc. de la partie *Je veux*.

Cadre IV-1. Extrait des sous-grammaires noyau et d'actions.

Grammaire noyau	*sentence=condition action {CONDITION:<1> ACTION:<2>}* *action= actionName [operator action]* *actionName=!actionGram!* *operator= »et «*
Grammaire d'actions	*actionName= open / close / watch / listen / switchOn / switchOff* *open= »ouvrir « opennable {LINK<1>}* *close= »fermer « closable {LINK<1>}* *watch= »regarder » observable « sur « viewer {<1>LINK<2>}* *listen= »ecouter « listenable {LINK<1>}* *switchOn= »allumer « switchOnAble {LINK<1>}* *switchOff= »eteindre « switchOffAble {LINK<1>}* *opennable=!abstractThingsGram!* *closable=!abstractThingsGram!* *observable=!abstractThingsGram!* *viewer=!abstractThingsGram!* *listenable=!abstractThingsGram!* *switchOnAble=!abstractThingsGram!* *switchOffAble=!abstractThingsGram!*

Rappelons que la construction de la représentation pivot est incrémentale en réaction immédiate aux choix de l'utilisateur final. Supposons que l'utilisateur choisisse l'élément *regarder*. EdProg colorie, dans la représentation pivot, le chemin correspondant à ce choix, en avertit l'Analyseur Lx-Sx qui poursuit la construction du sous-arbre de *watch*.

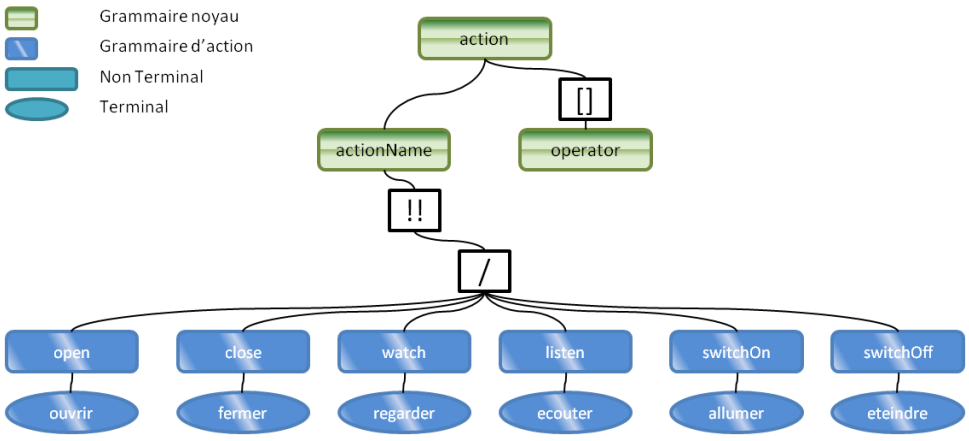


Figure A2-3. Sous-arbre des choix possibles de verbe d'action.

watch est une concaténation du terminal *regarder*, du non-terminal *observable*, du terminal *sur* et du non-terminal *viewer* (ignorons les décorations sémantiques pour l'instant). Il s'agit en effet d'une action qui consiste à regarder « quelque chose, qui est donc observable » et pour regarder cette chose, il faut un viewer, c'est-à-dire un dispositif qui permet de regarder. L'analyseur se doit d'identifier tous les observables possibles de l'espace intelligent, puis, dès que l'utilisateur a fait son choix d'observable, proposer tous les viewers capables de montrer l'observable d'intérêt. C'est là

qu'intervient l'ontologie et la BdC. Le non-terminal *observable* fait référence à la sous-grammaire des classes abstraites (Cadre A2-2).

Cadre A2-2. Extrait de la sous-grammaire des classes abstraites.

Grammaire des classes abstraites	observable=!abstractThingsBDC!

Dans la sous-grammaire des classes abstraites, la définition d'*observable*, qui redirige sur la BdC, se traduit par une requête SPARQL. Cette requête demande toutes les instances d'observables présentes dans l'espace intelligent ainsi que les noms de ces instances sous lesquels l'utilisateur final les désigne. Le cadre A2-3 montre, exprimée dans le formalisme ABNF, la réponse de la BdC à cette requête. L'analyseur Lx-Sx complète le sous-arbre selon le schéma de la figure A2.4, en avertit EdProg qui est alors en mesure de proposer un menu à deux éléments : *la température de l'eau*, *l'état de la machine à laver*.

Cadre A2-3. Sous-classes d'observable dans la BdC exprimés en ABNF.

BDC	observable=temperatureObservable / washingMachineStateObservable temperatureObservable="la temperature de l'eau" washingMachineStateObservable="l'état de la machine à laver"

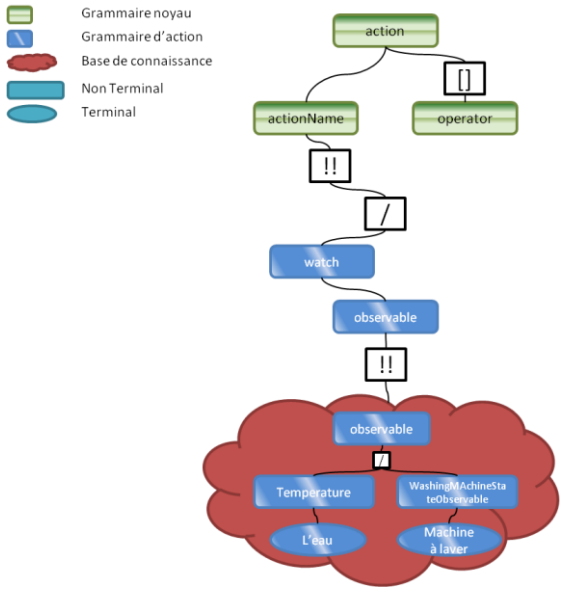


Figure A2.4. Sous-arbre *watch* résultant du choix du terminal *regarder* par l'utilisateur final.

En synthèse, le diagramme de séquence de la figure A2.5 explicite les interactions entre le gestionnaire de la représentation pivot de EdProg, l'analyseur Lx-Sx et la BdC.

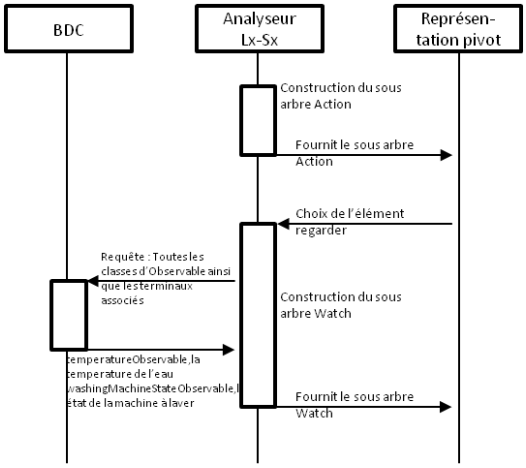


Figure A2.5. Diagramme de séquence UML des interactions entre le gestionnaire de la représentation pivot de EdProg, Analyseur Lx-Sx et la BdC.

10.4 L'éditeur de programme EdProg

L'éditeur de programme EdProg, rappelons-le, autorise l'utilisation de deux syntaxes concrètes : LPN et visuelle iconique. Comme le montre l'image écran de la figure 4, il comprend donc deux IHM (*IHM.LPN* et *IHM.Iconic*) et un noyau fonctionnel constitué de trois composants : le gestionnaire de la représentation pivot et un générateur pour chacune des deux syntaxes concrètes : *LPNGen* et *IconicGen*. Nous avons détaillé dans la section précédente les interactions du gestionnaire de la représentation pivot avec l'analyseur Lx-Sx. Nous complétons ici la description du fonctionnement d'EdProg. Le diagramme de séquence de la figure A2.6 montre les interactions entre les composants du noyau fonctionnel de EdProg, ses deux IHM et l'analyseur sémantique.

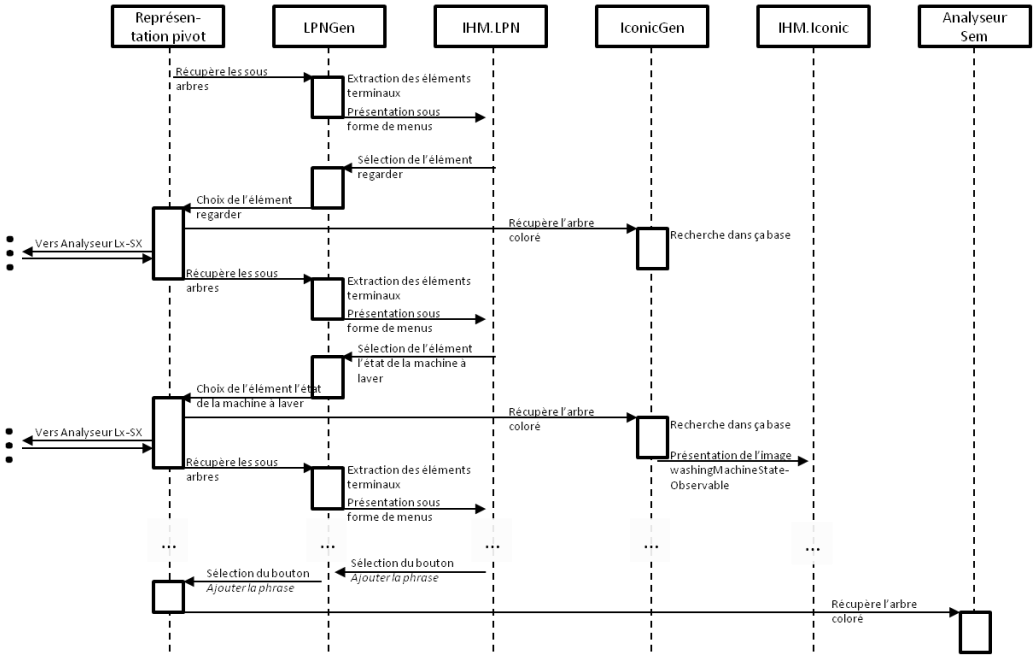


Figure A2.6. Diagramme de séquence UML des interactions entre le gestionnaire de la représentation pivot de EdProg, les générateurs LPNGen et IconicGen, IHM.LPN et IHM.Iconic et AnalyseurSem.

10.5 Générateur LPN et IHM.LPN

Le générateur LPN, *LPNGen*, sert d'intermédiaire entre *IHM.LPN* et le gestionnaire de la représentation pivot. Il récupère auprès du gestionnaire de la représentation pivot, les sous-arbres nouvellement produits par l'analyseur Sx-Lx, en extrait les éléments terminaux pour présentation sous forme de menus par *IHM.LPN*.

En effet, à chaque menu déroulant géré par *IHM.LPN* correspond, dans la représentation pivot, un sous-arbre des choix potentiels permis. *IHM.LPN* gère l'interaction avec l'utilisateur et, sur sélection d'un élément de menu, en avertit *LPNGen*. Ce dernier, qui maintient la correspondance entre les éléments de menu et les feuilles de la représentation pivot, notifie le gestionnaire de la représentation pivot du choix effectué. Cette notification a trois conséquences : le gestionnaire actualise le coloriage des chemins retenus par l'utilisateur, l'analyseur Lx-Sx peut poursuivre le parcours des sous-grammaires et, au nom de l'égale opportunité, *IconicGen* peut actualiser la présentation gérée par *IHM.Iconic*.

À titre illustratif, supposons que le sous-arbre *action* de la représentation pivot soit dans l'état représenté dans la figure A2.3. Alors, *IHM.LPN* présente le menu de la figure A2.7 et se met à l'écoute de l'utilisateur.

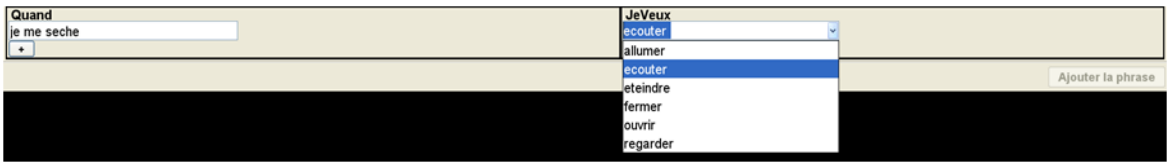


Figure A2.7. À droite, le menu des actions possibles correspondant au sous-arbre de la représentation pivot de la figure A2.3.

Supposons que l'utilisateur sélectionne l'élément *regarder*. L'élaboration de la représentation pivot reprend selon le processus décrit plus haut pour aboutir à l'état de la figure A2.4. L'IHM en LPN est actualisée dans l'état de la figure A2.8. Notons que le menu des actions reste disponible. Ainsi, l'utilisateur peut revenir sur ses décisions entraînant la réactualisation des coloriages de la représentation pivot et donc celle de l'état de l'IHM de l'éditeur. Notons aussi que la construction d'une phrase en LPN, dirigée par l'analyseur Lx-Sx, est nécessairement syntaxiquement correcte. De plus, l'analyseur faisant appel à la BdC, la phrase est nécessairement sémantiquement correcte.

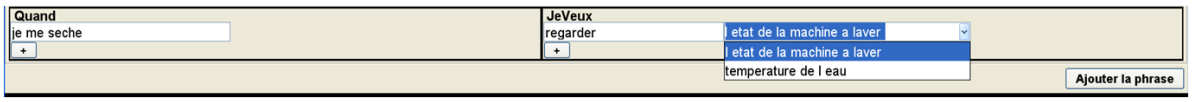


Figure A2.8. L'IHM de l'éditeur en LPN est actualisée avec l'apparition du menu des observables présents dans l'espace intelligent suite à la sélection de l'élément regarder du menu des actions.

Sur détection de la sélection du bouton *Ajouter la phrase*, *IHM.LPN* avertit, via le *LPNGen*, le gestionnaire de la représentation pivot que la phrase est complète et de là, le gestionnaire fournit l'arbre colorié à *AnalyseurSem*, l'analyseur sémantique que nous étudions en détail plus loin.

10.6 Générateur iconique et IHM.iconic

Dans leur version actuelle, le générateur iconique *IconicGen* et son IHM, *IHM.Iconic*, sont minimalistes (pour ne pas dire pauvrissimes). Leur présence sert à montrer comment l'architecture d'EdProg assure l'extensibilité de KISS vers du multisyntaxe avec égale opportunité totale.

IconicGen possède une base de couples <image, métadescription> renseignés manuellement au moyen d'un éditeur pour utilisateur averti (cf. figure A2.9). Les métadonnées sont issues de l'ontologie de la BdC. Dans l'exemple de la figure A2.9, l'image du dispositif de nom *Machine à laver* est métadécrite par le nom de la classe de ce dispositif : *washingMachineStateObservable*, sous-classe d'*Observable* dans la BdC.

Figure A2.9. Ajout dans la base de données de *IconicGen* de la représentation iconique du dispositif *Machine à laver* métadécrit par le nom de classe *washingMachineStateObservable* issu de la BdC.

Illustrons le fonctionnement de *IconicGen* et de son IHM en reprenant l'exemple de la figure A2.8 où l'utilisateur s'apprête à sélectionner *l'état de la machine à laver*. La représentation pivot correspondante est alors celle de la figure A2.4.

Sur sélection de l'élément *l'état de la machine à laver*, le processus décrit dans le diagramme de séquence UML de la figure A2.6 est appliqué : le gestionnaire de la représentation pivot, averti de la sélection, diffuse la notification à qui de droit, y compris à *IconicGen*. Ce dernier trouve le non-terminal *washingMachineStateObservable* commun à sa base et au sous-arbre modifié : il fournit à *IHM.Iconic* l'image associée (voir figure A2.10).

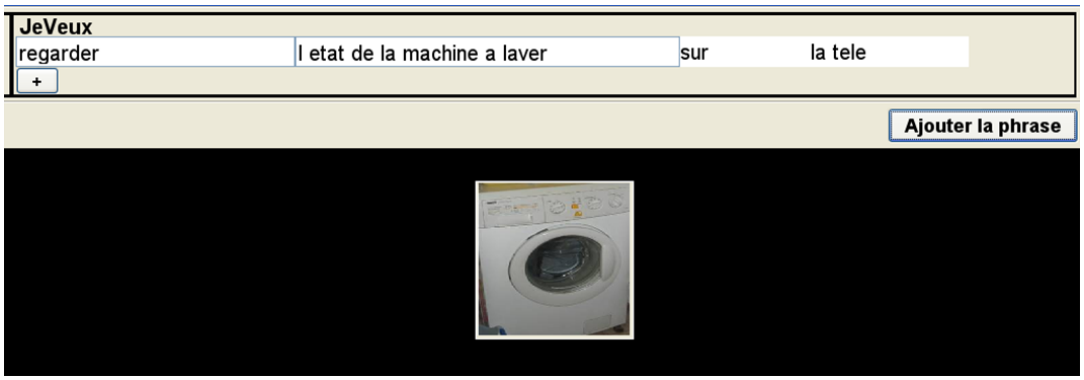


Figure A2-10. Sur sélection de l'élément de menu *l'etat de la machine à laver* dans l'IHM gérée par *IHM.LPN*, la photo du dispositif *Machine à laver* apparaît dans l'IHM de la partie iconique.

Toute phrase validée par l'utilisateur, on le rappelle, a pour effet de solliciter l'analyseur sémantique de KISS.

10.7 Analyseur sémantique et génération de code

Le rôle central de l'analyseur sémantique, *AnalyseurSem*, est de traiter les décorations sémantiques de l'arbre abstrait correspondant à la phrase que l'utilisateur final vient de valider. Ces traitements doivent faciliter la génération de code par *Générateur* dans le langage cible attendu par le gestionnaire de contexte.

10.7.1 Principes

AnalyseurSem utilise les opérateurs du cadre A2-4 que nous avons définis pour l'expression des traits sémantiques des sous-grammaires de KISS.

Cadre A2-4. Opérateurs du langage d'expression de la sémantique des non-terminaux des sous-grammaires de KISS.

CONDITION	Dénote la racine du sous-arbre *condition*
ACTION	Dénote la racine du sous-arbre *action*
:	Opérateur d'assignation
<i>	Désignation du i^{ème} non-terminal d'une règle de production
PC	Information complémentaire (par exemple, information de localisation) nécessaire à la génération des futurs Points de Coupe des AAs.
PARAM_STRING, PARAM_INT, PARAM_ BOOL	Informations sur les paramètres
EQUAL, SUP, INF	Opérateurs de condition
TOBOOL	Dénote la nécessité de tester l'égalité de ses deux arguments et, si c'est le cas, émettre « vrai »
LASTING	Dénote un aspect duratif (ou progressif)
INST	Dénote un aspect inchoatif (ou instantané)
LINK	Dénote une action de création de liaison

- La décoration CONDITION dénote, dans l'arbre abstrait de la phrase à traduire, la racine de l'arbre qui correspond à la description de la situation qui lorsqu'elle se présentera devra provoquer l'exécution d'un plan d'adaptation. Le nœud décoré de ACTION dénote l'arbre correspondant au plan d'adaptation.
- <i> permet de désigner des non-terminaux dans les règles de production des sous-grammaires. Par exemple, dans la règle *sentence=condition action {CONDITION :<1> ACTION :<2>}* de la sous-grammaire noyau, <1> désigne le premier non-terminal de la règle, (ici, *condition*) et ce non-terminal correspond à la description d'une situation (opérateur d'assignation « : »).
- PC désigne toute information complémentaire nécessaire à la génération de Point de Coupe dans les AAs. Par exemple, dans la règle de production de la grammaire de contexte, *sensorBedroomIn= ''dans la chambre'' {PC location :bedroom},* {PC location:bedroom} précise que les instances de sensorBedroomIn doivent se trouver en un lieu de classe bedroom.

- TOBOOL dénote la nécessité de tester, à l'exécution, l'égalité de ses deux arguments et si c'est le cas, émettre un événement « Vrai ». Par exemple, dans la règle de production *presenceDetectorIn= …/…/… {OccupancyStateTOBOOL''Occupied''}, {OccupancyStateTOBOOL''Occupied''}* signifie : si la variable d'état *OccupancyState* des capteurs de présence vaut *Occupied*, alors le booléen Vrai doit être émis.
- LINK exprime la création d'une connexion entre ses deux arguments. L'absence du 1er argument indique que l'action est inchoative. Elle est durative s'il y a deux arguments. Nous expliquons ces termes avec les opérateurs LASTING et INST.
- Les opérateurs LASTING et INST expriment les aspects *duratif*[13] et *inchoatif* de conditions ou d'actions. Expliquons la distinction par l'exemple, sans entrer dans les subtilités de la linguistique : la condition *quand je rentre dans la chambre* a un caractère instantané. Si une action doit être effectuée en réaction, celle-ci doit débuter dès l'entrée dans la chambre. L'aspect de cette condition est inchoatif. *A contrario*, la condition *quand je suis dans la chambre* a une durée. Si une action doit être effectuée en réaction, celle-ci doit avoir lieu tant que la chambre est occupée. L'aspect de la condition est alors duratif. De même, l'aspect de l'action *allumer la lumière* est inchoactif alors que celui de l'action *regarder la télévision* est duratif (s'il s'agit bien d'être en train de regarder la télévision, non pas juste jeter un regard au poste de télévision auquel cas, l'aspect serait inchoatif).

Il convient donc de considérer quatre combinaisons :

1. Condition et action duratives. Exemple de phrase type : *quand je suis dans la chambre, je veux regarder l'état de la machine à laver sur la télé.*
2. Condition et action inchoatives. Exemple : *quand je rentre dans la chambre, je veux allumer la lumière*.
3. Condition durative, action inchoative. Exemple : *quand je suis dans la chambre, je veux allumer la lumière*.
4. Condition inchoative, action durative. Exemple : *quand je rentre dans la pièce, je veux regarder l'état de la machine à laver sur la télé*.

A partir de ces combinaisons, l'analyseur sémantique doit piloter la génération de prédicats SPARQL et d'AA. Cette génération doit prendre en considération les éléments suivant : une condition inchoative se traduit dans un niveau d'adaptation réflexe, seul garant d'une réactivité conforme à cet aspect ; le greffon d'un AA définit toujours une liaison entre deux composants au minimum.

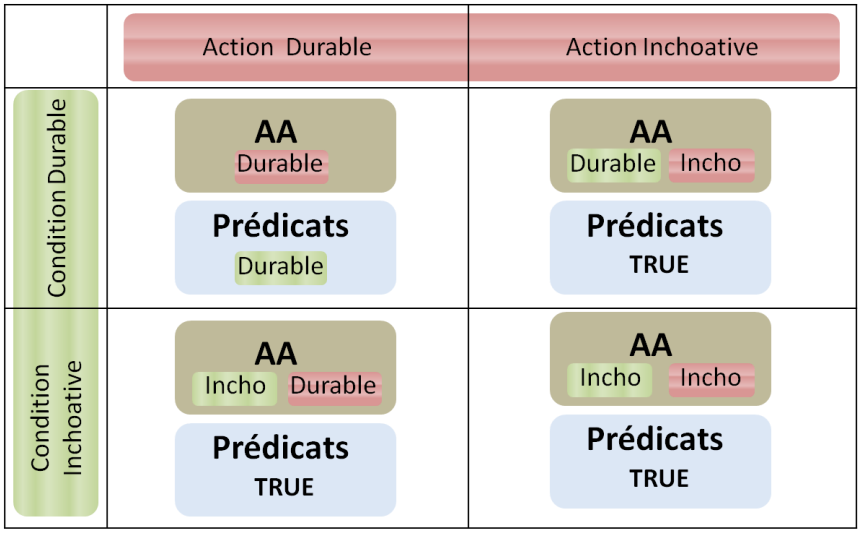
Par conséquent, la génération répond aux principes suivant (le tableau A2-2 résume ces principes) :

- Une action durative se traduit en AA. En effet, une telle action dispose de deux composants, critère suffisant pour définir un AA.
- Une condition inchoative s'ajoute à la partie gauche de l'AA à générer. Que l'action soit durative ou inchoative, cette condition doit faire partie d'un AA comme source d'évènement.
- Une action inchoative possède un unique composant, il est nécessaire de lui en fournir un second pour générer un AA. La condition quel que soit son aspect est alors utilisé comme source.
- Une condition durative, dans le cas où l'action est durative, se traduit en prédicats SPARQL.

[13] Les linguistes utilisent aussi le terme progressif.

- Dans tous les cas où il manque une condition pour générer un prédicat SPARQL, un prédicat TRUE est généré.

Tableau A2.3. Principes de génération de prédicats SPARQL et d'AA à partir de l'aspect de la condition et de l'aspect de l'action.



Nous illustrons la mise en application de ces principes par deux exemples, l'un où condition et action sont inchoatives, l'autre où condition et action sont duratives.

10.7.2 Exemple 1 : condition et actions inchoatives

L'utilisateur a rédigé la phrase : *Quand je rentre dans la chambre, je veux allumer la lumière de la chambre.* AnalyseurSem produit l'expression sémantique du cadre A2.5. Cette expression sémantique va permettre au générateur de produire l'AA du cadre A2.7 ainsi que les deux prédicats SPARQL du cadre A2.6 qui spécifient la situation dans laquelle cet AA doit être déployé. Le couple <Situation-AA> est ensuite transmis au Gestionnaire de Contexte au format XML.

Cadre A2.5. Expression sémantique produite par AnalyseurSem pour la phrase *Quand je rentre dans la chambre, je veux allumer la lumière de la chambre* : Les deux dernières lignes spécifient que les équipements de type *switchOnAbleLight* et *presenceDetectorIn* dont il est question sont ceux dont la localisation est *bedroom*. La 1ère ligne signifie : produire un événement lorsque l'événement *OccupancyState* de tous les équipements de type *presenceDetectorIn*, vaut *Occupied* et relier cet événement (opérateur *LINK*) à la méthode *SetTarget* (avec le paramètre *True*) de tous les équipements de type *switchOnAbleLight.*

presenceDetectorIn OccupancyStateTOBOOL"Occupied"LINKswitchOnAbleLight "SetTarget" PARAM_BOOL "True"

PC location:bedroom switchOnAbleLight

PC location:bedroom presenceDetectorIn

Cadre A2.6. Ces deux prédicats exprimés en SPARQL spécifient la situation dans laquelle l'AA du cadre IV-7 devra être déployé. À gauche, le lieu d'exécution de la mise en œuvre du AA (monde réel ou monde virtuel du simulateur). À droite, *True* indique que l'AA est toujours valide.

```
ASK ?h WHERE
?x type ModeSelector
?x Mode ?h
FILTRE (?h=real)
```

```
True
```

Cadre A2.7. AA généré correspondant à l'expression sémantique du cadre IV-5. Trouver tous les équipements de type *presenceDetectorIn*, de localisation *bedroom* et présents dans le *monde réel*

(*virtual=false*) ; de même, trouver tous les équipements de type *switchOnAbleLight*, de localisation *bedroom*, et présents dans le *monde réel*. Si ces deux ensembles, dénotés respectivement par les variables *presenceDetectorIn* et *swicthOnAbleLight*, ne sont pas vides, alors appliquer l'*advice* de nom *presenceDetectorIn_falseswitchOnAbleLight_11* avec les deux paramètres *presenceDetectorIn* et *switchOnAbleLight*). Cet advice consiste à instancier le composant *bool0*, instance de *WComp.BasicBeans.StringComparator*, un comparateur de chaîne disponible dans WCOMP avec *Occupied* comme valeur de référence. Pour tout élément de l'ensemble *presenceDetectorIn*, (1) créer une connexion entre l'événement *OccupancyState_Evented_NewValue* produit par cet élément et la méthode *Compare* de *bool0*, (2) créer une connexion entre l'événement *MatchedBoolean* produit par *bool0* (si le résultat de la comparaison est vrai), et la méthode *SetTarget* de tous les éléments de *swicthOnAbleLight*. Ce qui a bien pour effet d'allumer la lumière si la chambre est occupée.

```
presenceDetectorIn = *?type=presenceDetectorIn&location=bedroom&virtual=false
switchOnAbleLight  = *?type=switchOnAbleLight&location=bedroom&virtual=false
advice presenceDetectorIn_falseswitchOnAbleLight_11(presenceDetectorIn, switchOnAbleLight) :
bool0:WComp.BasicBeans.StringComparator(ReferenceValue:=”Occupied”)
presenceDetectorIn.^OccupancyState_Evented_NewValue -> (bool0.Compare)
bool0.^MatchedBoolean -> (switchOnAbleLight. SetTarget )
```

Nous allons maintenant décrire comment *AnalyseurSem* aboutit à l'expression sémantique du cadre A2.5. Il faut pour cela partir de l'arbre abstrait de la phrase source que EdProg a transmis à AnalyseurSem (voir figure A2.11). Il faut également exploiter les traits sémantiques des non-terminaux de l'arbre au moyen des sous-grammaires de KISS (cadres A2.8, A2.9, A2.10).

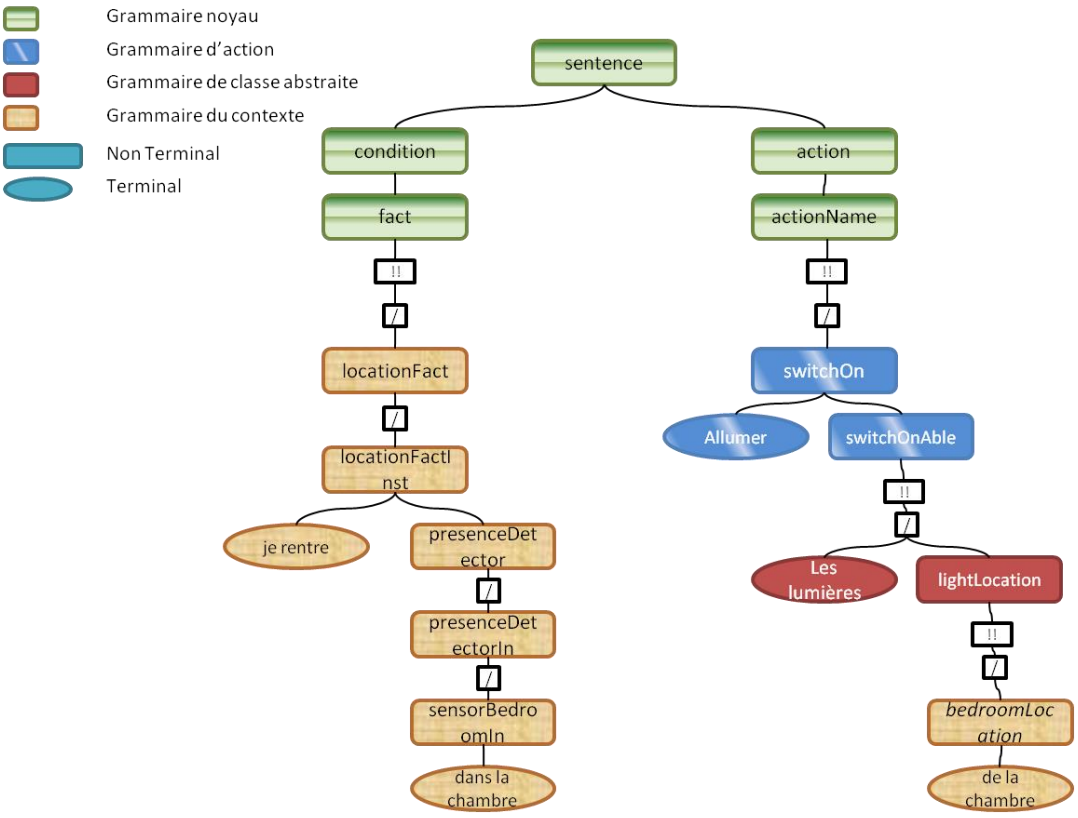


Figure A2.11. Arbre abstrait correspondant à la phrase : *quand je rentre dans la chambre, je veux allumer la lumière de la chambre*.

Cadre A2.8. Extrait de la grammaire du contexte.

Grammaire du contexte	fact=locationFact / timeFact locationFact=locationFactLasting / locationFactInst locationFactInst= »je rentre « presenceDetector {INST<1>} presenceDetector=presenceDetectorIn / presenceDetectorOut presenceDetectorIn= sensorBathroomIn / sensorBedroomIn / sensorKitchenIn {OccupancyStateTOBOOL »Occupied »} sensorBedroomIn= »dans la chambre » {PC location:bedroom}

Cadre A2.9. Extrait de la grammaire des classes abstraites.

Grammaire de classes abstraites	*switchOnAble=switchOnAbleLight / switchOnAbleCoffee* switchOnAbleLight= »les lumieres « [lightLocation] [lightIntensity] { »SetTarget » PARAM_BOOL « True »} lightLocation=!contextualInfoGram!

Cadre A2.10. Extrait de la grammaire du contexte pour le non-terminal lightLocation

Grammaire du contexte	*lightLocation=bathroomLocation / bedroomLocation / kitchenLocation* bathroomLocation= »de la salle de bain « {PC location:bathroom} bedroomLocation= »de la chambre « {PC location:bedroom} kitchenLocation= »de la cuisine « {PC location:kitchen}

AnalyseurSem applique un algorithme à 3 étapes, d'abord pour la partie *condition*, puis pour la partie *action* de l'arbre : (1) recherche de la sémantique générale de la partie en cours d'analyse et de son aspect inchoatif/duratif, (2) recherche de traits complémentaires (avec les opérateurs PC pour des informations complémentaires sur des objets, PARAM_STRING, PARAM_INT et PARAM_BOOL pour les informations sur les paramètres), (3) recherche des ports pour la création de futures connexions.

Commençons par le sous-arbre *condition* de la Figure A2.11, *quand je rentre dans la chambre*.

étape 1. Le non-terminal *locationFactInst* donne la réponse à la première étape : dans la grammaire du cadre A2.8, la règle de production *locationFactInst=''je rentre'' presenceDetector {INST <1>}* , {INST <1>} indique que l'on a affaire à une condition inchoative. Le résultat de cette 1ère étape est :

INSTpresenceDetectorIn

étape 2. AnalyseurSem cherche ensuite à acquérir les informations complémentaires (opérateur PC) pour chaque non-terminal du sous-arbre *condition* et cela en relation avec les sous-grammaires. Seul le non-terminal *sensorBedroom* possède un trait sémantique : *{PC location:bedroom}*. Le résultat de la 2ème étape est maintenant :

INSTpresenceDetectorIn

PC location:bedroom presenceDetectorIn

étape 3. Cette étape consiste à trouver les ports pour établir les futures connexions. AnalyseurSem procède comme précédemment à parcourir les non-terminaux de l'arbre abstrait et les grammaires, mais cette fois en repérant les ports. Seul le non-terminal *presenceDetectorIn* répond à la recherche avec *OccupancyStateTOBOOL"Occupied"*. L'analyse de la partie condition de la phrase est maintenant complète :

INSTpresenceDetectorIn OccupancyStateTOBOOL"Occupied"

PC location:bedroom presenceDetectorIn

Le même algorithme à trois phases (sémantique générale, informations complémentaires, ports) est appliqué à la partie *action* de la Figure A2.11, *je veux allumer la lumière de la chambre*.

étape 1. Cette étape s'appuie sur le non-terminal *switchOn* (cf. la grammaire du cadre A2.1). On obtient :

LINKswitchOnAbleLight

étape 2. Après la recherche des informations PC, le résultat de la 2ème étape produit :

LINKswitchOnAbleLight

PC location:bedroom switchOnAbleLight

étape 3. Dans la sémantique déjà extraite, il n'y a qu'un seul non-terminal impliqué, donc un seul port de communication.

LINKswitchOnAbleLight "SetTarget" PARAM_BOOL "True"

PC location:bedroom switchOnAbleLight

L'opérateur *PARAM_BOOL* a deux arguments : l'argument de gauche désigne un nom de méthode à un paramètre booléan ; l'argument de droite est la valeur du paramètre d'appel de cette méthode. Ici la méthode est *SetTarget,* elle prendra comme paramètre *True*.

Nous constatons que l'opérateur LINK n'a pas de source, ce qui est contraire à la sémantique d'une liaison. Dans ce cas, l'analyseur sémantique utilise l'expression sémantique de la partie condition comme source de l'opérateur LINK alors que cette partie condition devrait servir à la description de la situation. La situation devient true (partie droite du cadre A2.6), sa description devenant une source événementielle d'AA. On obtient au final :

presenceDetectorIn OccupancyStateTOBOOL"Occupied"LINKswitchOnAbleLight "SetTarget" PARAM_BOOL "True"

PC location:bedroom switchOnAbleLight

PC location:bedroom presenceDetectorIn

10.7.3 Exemple 2 : condition et actions duratives

L'utilisateur a rédigé la phrase : *Quand je suis dans la chambre, je veux regarder l'état de la machine à laver sur la télé.* AnalyseurSem produit l'expression sémantique du cadre A2.11 qui aide le générateur à produire l'AA du cadre A2.13 ainsi que les deux prédicats SPARQL du cadre A2.12 qui spécifient la situation dans laquelle cet AA doit être déployé.

Cadre IV-A2.11. Expression sémantique produite par AnalyseurSem pour la phrase *Quand je suis dans la chambre, je veux regarder l'état de la machine à laver sur la télé* : La 1ère ligne décrit la situation qui nécessitera une adaptation. La 2ème ligne spécifie que les *presenceDetectorIn* dont il est question doivent se trouver dans une chambre. Ces deux lignes conduisent à la génération du prédicat de droite du cadre A2.12. La dernière ligne de l'expression sémantique se lit comme suit : pour tout équipement qui déclare dans ses métadonnées être un *washingMachineObservable*, écouter sa variable événementiel *RemainingTime.* Quand cette variable change, la transférer aux équipements qui déclarent dans leurs métadonnées être des *display* via leur méthode *setValue.* Cette ligne permettra de générer l'advice de l'AA du cadre A2.12.

```
LASTINGpresenceDetectorIn OccupancyStateTOBOOL"Occupied"
PC location:bedroom presenceDetectorIn
washingMachineObservable RemainingTime LINKdisplay SetValue
```

Cadre A2.12. les deux prédicats exprimés en SPARQL spécifient la situation dans laquelle l'AA du cadre A2.13 devra être déployé. À gauche, le lieu d'exécution de la mise en œuvre du AA (ici, monde réel). À droite, la chambre doit être occupée.

```
ASK ?h WHERE
?x type ModeSelector
?x Mode ?h
FILTRE (?h=real)
```

```
ASK ?h WHERE
?x type presenceDetector
?x location bedroom
?x OccupancyState ?h
FILTRE (?h=Occupied)
```

Cadre A2.13. AA généré correspondant à l'expression sémantique du cadre A2.11. Trouver tous les équipements de type *washingMachineObservable* et présents dans le *monde réel* (*virtual=false*) ; de même, trouver tous les équipements de type *display* et présents dans le *monde réel*. Si ces deux ensembles, dénotés respectivement par les variables *washingMachineObservable* et *display*, ne sont pas

vides, alors appliquer l'*advice* de nom *washingMachineObservable_falsedisplay_11* avec les deux paramètres *washingMachineObservable* et *display*. Cet advice consiste à créer pour tout élément de *washingMachineObservable* une connexion entre l'événement *RemainingTime_Evented_NewValue* produit par cet élément et la méthode *SetValue* de tous les éléments de *display*.

```
washingMachineObservable = *?type=washingMachineObservable&virtual=false
display = *?type=display&virtual=false
advice washingMachineObservable_falsedisplay_11(washingMachineObservable, display) :
washingMachineObservable.^RemainingTime_Evented_NewValue -> (display.SetValue)
```

Comme précédemment, *AnalyseurSem* aboutit à l'expression sémantique du cadre A2.11 à partir de l'arbre abstrait de la phrase source (Figure A2.12) et en s'appuyant sur les traits sémantiques des non-terminaux au moyen des sous-grammaires de KISS (cadres A2.14, A2.15, A2.16).

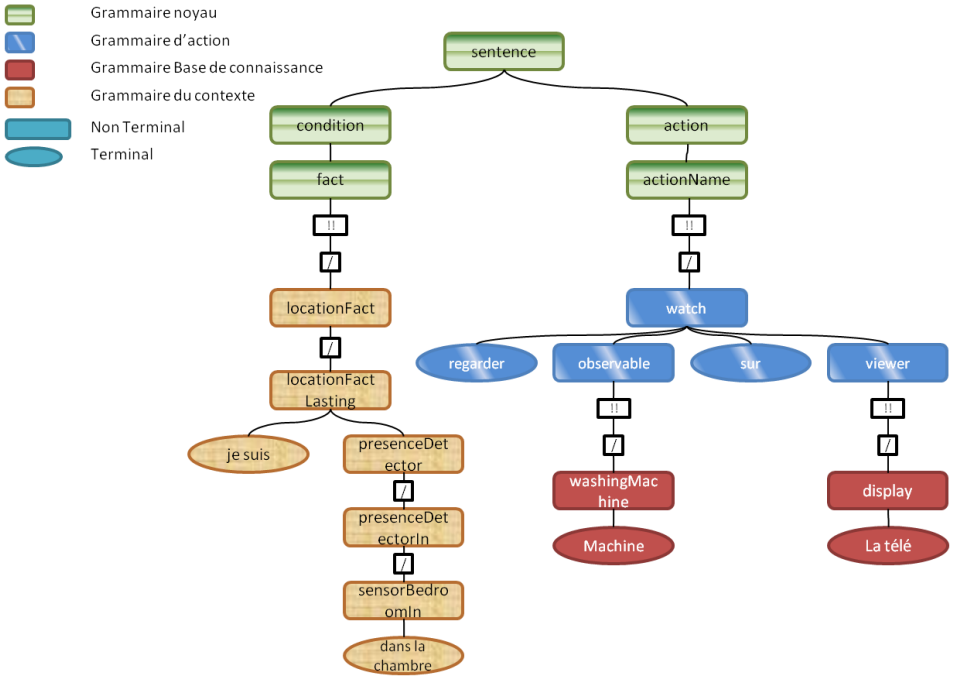


Figure A2.12. Arbre abstrait correspondant à la phrase *quand je suis dans la chambre, je veux regarder l'état de la machine à laver sur la télé*.

Analyse sémantique de la partie *condition* de l'arbre abstrait de la Figure A2.12, *quand je suis dans la chambre.*

étape 1. Le non-terminal concerné est *locationFactLasting*, dont l'aspect, d'après la règle de production de la grammaire de contexte (cadre A2.14) est duratif (*LocationFactLasting="je suis" presenceDetector {LASTING<1>*}. Le résultat est donc :

LASTINGpresenceDetectorIn

étape 2. Recherche des informations complémentaires. La règle de production *sensorBedroomIn="dans la chambre" {PC location:bedroom}* (cadre A2.14) conduit au résultat suivant :

LASTINGpresenceDetectorIn

PC location:bedroom presenceDetectorIn

étape 3. Recherche des ports. La grammaire du cadre A2.14 nous offre une seule proposition avec la règle de production *presenceDetectorIn= sensorBathroomIn / sensorBedroomIn / sensorKitchenIn {OccupancyStateTOBOOL"Occupied"}* ce qui donne au final pour la partie condition :

LASTINGpresenceDetectorIn OccupancyStateTOBOOL"Occupied"

PC location:bedroom presenceDetectorIn

En français, cette expression s'écrit : pour tous les équipements qui déclarent dans leur métadonnées être des *presenceDetectorIn* et se trouver dans *bedroom*, écouter leur variable événementiel *OccupancyState.* Quand elle vaut *Occupied*, émettre *vrai*.

Cadre A2.14. Extrait de la grammaire du contexte.

Grammaire du contexte	fact=locationFact / timeFact locationFact=locationFactLasting / locationFactInst locationFactLasting= »je suis « presenceDetector {LASTING<1>} presenceDetector=presenceDetectorIn / presenceDetectorOut presenceDetectorIn= sensorBathroomIn / sensorBedroomIn / sensorKitchenIn {OccupancyStateTOBOOL »Occupied »} sensorBedroomIn= »dans la chambre » {PC location:bedroom}

Analyse sémantique de la partie *action* de l'arbre abstrait de la Figure A2.12, *je veux regarder l'état de la machine à laver sur la télé*. Nous allons nous servir des extraits des grammaires des cadres A2.15 et A2.16.

Cadre A2.15. Extrait de la grammaire BdC.

Grammaire BdC	observable=temperatureObservable / washingMachineStateObservable temperatureObservable="la temperature de l'eau" {CurTemp} washingMachineStateObservable="l'état de la machine à laver" {RemainingTime}

Cadre A2.16. Extrait de la grammaire des sous-classes abstraites.

Grammaire des classes abstraites	viewer=display display="la tele " {SetValue}

étape 1. Le non-terminal concerné est *watch* défini par la règle de production du cadre A2.1 : *watch="regarder" observable "sur " viewer {<1>LINK<2>})*. Le résultat est donc :

washingMachineObservableLINKdisplay

étape 2. Recherche des informations complémentaires (opérateurs PC, PARAM_INT, PARAM_STRING, PARAM_BOOL). Les sous-grammaires n'indiquent aucune information complémentaire. Le résultat reste :

washingMachineObservableLINKdisplay

étape 3. Recherche des ports des entités washingMachineObservable et display entre lesquels établir la liaison. Pour *washingMachineObservable*, il faut demander à la BdC (cf. cadre A2.15), pour *display*, l'information se trouve spécifié dans la grammaire des sous-classes abstraites (cadre A2.16). Ce qui donne au final pour la partie action :

washingMachineObservable RemainingTime LINKdisplay SetValue

Ce qui se lit en français : Pour tous les équipements qui déclarent dans leur métadonnées être des *washingMachineObservable*, écouter leur variable événementiel *RemainingTime.* Quand cette variable change, la transférer aux équipements qui déclarent dans leurs métadonnées être des *display* via leur méthode *setValue*.

10.8 Simulateur, metteur au point

Le monde simulé permet à l'utilisateur de vérifier que le programme qu'il vient de concevoir est conforme à ses attentes, avancer ou reculer dans le temps à la vitesse désirée. Le moteur du monde simulé doit :

- Fournir une représentation virtuelle de l'environnement réel dans lequel évolue l'utilisateur.
- Permettre à l'utilisateur de déplacer un avatar qui le représente virtuellement.
- Permettre l'ajout dynamique d'équipements virtuels existence duale des équipements physiques du monde réel.
- Permettre à l'utilisateur d'interagir avec les équipements du monde simulé.

Le tableau A2.3 reprend les différents critères de choix. Pour la représentation virtuelle 3D, notre choix s'est porté sur le moteur de jeu Blender. Succinctement, Blender permet d'éditer directement le monde simulé dans son moteur de rendu. La communauté utilisateurs est très active, ce qui facilite l'effort de développement, et surtout, Blender utilise des scriptes Python qui permettent notamment d'ajouter facilement un mécanisme de communication.

Sur le plan mise en œuvre, un équipement réel ou simulé est un service UPnP. La seule distinction parmi ces dispositifs réside dans leurs métadonnées. Un dispositif virtuel aura dans ses métadonnées : *virtual=true* alors que pour un dispositif du monde réel *virtual=false*.

Tableau A2.3. Comparaison de différents moteurs de monde virtuel.

	DiaSuite	OpenSim	Blender	jMonkeyEngine
Dimension maximale supportée	2D	3D	**3D**	3D
Qualité de la documentation du code source	Code source non fourni	Insuffisante	Faible	Médiocre
Codé en	Java	C Sharp	Python	Java
Langage de scripting	Java	LSL	**Python**	Java
Doc. pour l'utilisation du logiciel/API	Bonne	Médiocre	**Bonne**	Bonne
Editeur de modèles 3D	Non	Non	**Oui**	Non
Moteur physique	Non	Oui	Oui	Oui
Pilotage de l'extérieur	Facile	Requiert la modification du code source	Possible	Facile
Séparation des entités	Oui	Oui	Oui	Oui
Relation entre les entités	Oui	Oui	Oui	Oui
Importation de modèles 3D et des animations	Non	Limité	**Oui**	Oui
Communauté	Inexistante	Faible	Active	Active
Définition et animation des gestes	Non	Non	**Oui**	Non
Extensible	Non	Non	**GUI et moteur de jeu extensible**	Probablement oui

Le monde simulé peut également être utilisé comme moyen de piloter le monde réel. Dans ce cas, les équipements présents dans le monde simulé sont directement connectés avec leurs homologues

réels. Ce mode de connexion se passe de l'infrastructure cible, et ne rentre pas dans le cadre de la programmation d'habitat intelligent par l'utilisateur.

10.9 Grammaire complète du scénario

Meta-Grammaire

ABNF
+ { *sémantique* } ; utilisé pour la création d'AA
+ ! *cross Référence* ! ; utilisé pour pointer sur une autre grammaire ou sur la BDC
+ ? *dépendance* ; utilisé pour demander des informations supplémentaire dans la sémantique

Sémantique :
CONDITION: localisation dans la grammaire de la structure condition
ACTION: localisation dans la grammaire de la structure action
<1> : index du 1er NT dans la grammaire
LINK : si LINK n'est pas précédé d'un index, l'action est instantanée, sinon elle est durable.
PC : ajoute une MetaData spécifique dans le point de coupe généré
PARAM_STRING : permet d'envoyer des paramètres de type string, les composants d'un param doivent être décoré par des " .
PARAM_STRING "a" : la sémantique du niveau supérieur donne la méthode qui convient avec ce paramètre
"a" PARAM_STRING "b" : a est la méthode qui prend en paramètre b
PARAM_INT : idem, mais avec des paramètres int.
PARAM_BOOL: idem, mais avec des paramètres boolean.
/ : correspond à l'alternation de la grammaire.
INST : indique que la condition est instantanée
LASTING : indique que la condition est durable
EQUAL, SUP, INF : des operateurs de condition
TOBOOL : operateur de transformation event to boolean.
X-X : séparateur de condition sémantique

Grammaire noyau : coreGramBobDomusV1.grm

sentence=condition action {CONDITION:<1> ACTION:<2>}
condition=fact [operator condition]
action=unitaryAction [operator action]
operator="et "

unitaryAction=actionName

fact=!contextualInfoGram!
actionName=!actionGram!

Grammaire d'action : actionGramBobDomusV1.grm

actionName=wakeUp / open / close / listen / switchOn / switchOff / watch / heatUp

wakeUp="être réveillé avec " alarm {LINK<1>}
open="ouvrir " opennable {LINK<1>}
close="fermer " closable {LINK<1>}
listen="ecouter " listenable {LINK<1>}
switchOn="allumer " switchOnAble {LINK<1>}
switchOff="eteindre " switchOffAble {LINK<1>}
watch="watch" observable "sur " viewer {<1>LINK<2>}
heatUp="chauffer " heatAble {LINK<1>}

alarm=!abstractThingsGram!
opennable=!abstractThingsGram!
closable=!abstractThingsGram!
listenable=!abstractThingsGram!
switchOnAble=!abstractThingsGram!
switchOffAble=!abstractThingsGram!
observable=!abstractThingsGram!
viewer=!abstractThingsGram!
heatAble=!abstractThingsGram!

Grammaire de classe abstraite : abstractThingsGramBobDomusV1.grm

alarm=lightAlarm/audioAlarm
lightAlarm=switchOnAbleLight
audioAlarm="une musique " soundLevelParam switchOnAbleAudio {"SetVolume" PARAM_INT "0"}

switchOnAbleLight=("les lumieres " (lightLocation / lightIntensityLow / lightIntensityMedium / lightIntensityStrong) / (lightLocation lightIntensityLow / lightIntensityNull / lightIntensityMedium / lightIntensityStrong)) {"SetTarget" PARAM_BOOL "True"}
lightIntensityLow="avec une intensite faible " {"SetLoadLevelTarget" PARAM_INT "20"}
lightIntensityMedium="avec une intensite moyenne " { "SetLoadLevelTarget" PARAM_INT "50"}
lightIntensityStrong="avec une intensite forte " {"SetLoadLevelTarget" PARAM_INT "90"}
lightIntensityNull="avec une intensite null " {"SetLoadLevelTarget" PARAM_INT "0"}

soundLevelParam="a volume " soundLevelLow / soundLevelNormal / soundLevelStrong {"SetVolume" PARAM_STRING "Master"}
soundLevelLow="faible " {"SetVolume" PARAM_INT "50"}
soundLevelNormal="normal " {"SetVolume" PARAM_INT "65"}
soundLevelStrong="fort" {"SetVolume" PARAM_INT "85"}
switchOnAbleAudio=paramAudioMuteFalse0 paramAudioMute0 paramAudioMuteFalse1 {"SetMute" PARAM_INT "0"}
paramAudioMuteFalse0=" " {"SetMute" PARAM_INT "0"}
paramAudioMute0=" " {"SetMute" PARAM_STRING "Master"}
paramAudioMuteFalse1=" " {"SetMute" PARAM_BOOL "False"}

opennable="les volets " {"SetPosition" PARAM_STRING "Opened"}
closable="les volets " {"SetPosition" PARAM_STRING "Closed"}
listenable=audioAlarm

switchOnAble=switchOnAbleLight / switchOnAbleCoffe
switchOnAbleCoffe="la cafetiere " {"SetTarget" PARAM_BOOL "True"}

switchOffAble=switchOffAbleLight / switchOffAbleCoffe / switchOffAbleAudio {}
switchOffAbleLight=("la lumiere " [lightLocation]) {"SetTarget" PARAM_BOOL "False"}
switchOffAbleCoffe="la cafetiere " coffeMachine {"SetTarget" PARAM_BOOL "False"}
switchOffAbleAudio="la musique " paramAudioMute0 paramAudioMute1 {"SetMute" PARAM_INT "0"}
paramAudioMute1=" " {"SetMute" PARAM_BOOL "True"}

observable=temperatureObservable
viewer=temperatureDisplay
temperatureObservable="temperature de l eau " {CurState}
temperatureDisplay="la douche " {SetValue}
heatAble="l eau de la douche " {"SetTargetTemp" PARAM_INT "25"}

LightLocation=!contextualInfoGram!
coffeMachine=!contextualInfoGram!

Grammaire du contexte : contextualInfoGramBobDomusV1.grm

fact=locationFact / timeFact / dryingFact / buttonPress / buttonOn
locationFact=locationFactLasting / locationFactInst
timeFact="il est " time
dryingFact=drying {INST<1>}
buttonPress="j'appuie sur "button {INST<1>}
buttonOn=button "a ete enclenche " {LASTING<1>}

locationFactLasting="je suis " presenceDetector {LASTING<1>}
locationFactInst=locationFactInstIn / locationFactInstOut
locationFactInstOut="je sort " presenceDetectorInstOut {INST<1>}
locationFactInstIn="je rentre " presenceDetectorInstIn {INST<1>}

presenceDetector=presenceDetectorIn / presenceDetectorOut
presenceDetectorIn=sensorLitIn / sensorBathroomIn / sensorBedroomIn / sensorKitchenIn {OccupancyStateTOBOOL"Occupied"}
presenceDetectorOut=sensorLitOut / sensorBathroomOut / sensorBedroomOut / sensorKitchenOut {OccupancyStateTOBOOL"Unoccupied"}

sensorLitIn="dans le lit" {PC location:bed}
sensorLitOut="hors du lit" {PC location:bed}
sensorBathroomIn="dans la salle de bain" {PC location:bathroom}
sensorBathroomOut="hors de la salle de bain" {PC location:bathroom}
sensorKitchenIn="dans la cuisine" {PC location:kitchen}
sensorKitchenOut="hors de la cuisine" {PC location:kitchen}
sensorBedroomIn="dans la chambre" {PC location:bedroom}
sensorBedroomOut="hors de la chambre" {PC location:bedroom}

presenceDetectorInstIn=sensorLitIn / sensorBathroomIn / sensorBedroomIn / sensorKitchenIn {OccupancyStateTOBOOL"Occupied"}
presenceDetectorInstOut=sensorLitInstOut / sensorBathroomInstOut / sensorBedroomInstOut / sensorKitchenInstOut {OccupancyStateTOBOOL"Unoccupied"}

sensorLitInstOut="du lit" {PC location:bed}
sensorBathroomInstOut="de la salle de bain" {PC location:bathroom}
sensorKitchenInstOut="de la cuisine" {PC location:kitchen}
sensorBedroomInstOut="de la chambre" {PC location:bedroom}

time=clockTime/betweenTime
clockTime=clock {INSTTimeEQUAL<1>}
betweenTime="entre " clock "et " clock {INSTTimeSUP<1>X-XTimeINF<2>}
clock="21h" / "22h" / "07h05" / "09h" / "07h30" / "07h10" / "20h" / "10h" / "08h" / "07h" {2100/2200/705/900/730/710/2000/1000/800/700}
button="le bouton " {ValueTOBOOL"0"}

drying="je me seche" dryingTowel {OccupancyStateTOBOOL"Occupied"}
dryingTowel=" " {PC location:towel}

lightLocation=bathroomLocation / bedroomLocation / kitchenLocation
coffeMachine=" "

11 Annexe 3 – Déroulement complet de l'expérimentation

Accueil et signature du consentement

Bonjour,

Tout d'abord merci d'avoir bien voulu participer à notre expérience. Nous allons vous faire tester une application qui permet de piloter un habitat intelligent. C'est-à-dire un habitat dans lequel vous allez pouvoir programmer différents objets. Notre application est encore en phase d'évaluation et c'est pour cela que nous avons besoin de vos retours aussi bien positifs que négatifs. Tous vos commentaires sont importants pour que nous puissions améliorer l'application.

Afin de pouvoir exploiter les résultats, nous allons enregistrer ce que vous ferez sur l'ordinateur, nous allons vous enregistrer et vous filmer. Ces données sont uniquement à destination de la recherche et seront traitées de manière anonyme puis détruites une fois réalisée l'analyse de ces données (dans les 3 mois). Pour être conforme à la législation, nous vous demandons, si vous l'acceptez, de signer ce consentement pour enregistrer ces données expérimentales.

EXPERIMENTATEUR : FAIRE SIGNER LE CONSENTEMENT

Faire visiter l'appartement

DONNEES RECUEILLIES : VIDEO ET AUDIO

Nous allons maintenant nous rendre dans l'appartement « DOMUS ». Je vais vous faire visiter cet appartement et vous montrer les différents capteurs qu'il possède et les services qu'il offre. Tout d'abord, les pièces sont munies de capteurs de présence, c'est-à-dire que le système sait dans quelle pièce vous vous trouvez.

EXPERIMENTATEUR : donner au participant un plan de l'appartement avec tous les capteurs et effecteurs.

Dans la cuisine, vous avez des capteurs de présence, les volets sont programmables. Les lumières et la musique sont également programmables dans cette pièce. La cafetière peut aussi se déclencher de manière automatique selon les moments de la journée.

Dans la chambre, le lit est équipé d'un capteur qui détecte votre présence dès que vous vous êtes assis dessus. Il y a aussi un capteur qui vous détecte dans la pièce. Les volets de la chambre sont programmables. Comme dans la cuisine, les lumières et la musique sont programmables.

Sur la table de nuit, vous remarquez ce bouton qui permet d'enclencher certaines fonctions. En particulier, le matin il peut jouer le rôle de « réveil matin de la maison » et lancer un ensemble de tâches routinières qui se passent le matin : ouvrir les volets, chauffer l'eau de la douche à la bonne température, allumer la cafetière, …

Dans la salle de bain, Il y a également un capteur de présence général et un capteur qui détecte votre présence dans la douche. La température de l'eau de la douche s'affiche sur le mur. Les lumières sont programmables. Et le pose-serviette est assez surprenant car il détecte la présence ou non d'une serviette.

Dans le bureau, il y a également certaines fonctionnalités mais nous les avons désactivées de manière à ne pas perturber l'expérience.

Avez-vous des questions à nous poser concernant les capteurs ou les capacités de cet appartement ?

Présenter et faire tester l'application en langage naturel

DONNEES RECUEILLIES: VIDEO ET AUDIO

EXPERIMENTATEUR explique le fonctionnement de l'application, il fournit à l'utilisateur un guide d'utilisation simple de l'application.

EXPERIMENTATEUR fait un premier test avec la phrase « Allumer une lampe quand je suis dans la salle de bain », montrer le résultat en jouant le scénario dans Domus.

Maintenant, vous allez faire un exercice pour vous familiariser avec Domus en écrivant la phrase « Allumer une lampe quand je suis dans la salle de bain entre 20h et 22h ».

A vous ….

Nous allons aller voir ce qui se passe dans la salle de bain.

EXPERIMENTATEUR : constater que rien ne se produit car nous ne sommes pas dans le bon créneau horaire et montrer le fonctionnement de l'horloge virtuelle.

Faire modifier le créneau horaire par le participant.

Ok, donc nous allons voir si Domus a compris (retourner dans la salle de bains et refaire le test)

Présenter et faire tester DOMUS virtuel en 3D

DONNEES RECUEILLIES: VIDEO ET AUDIO

EXPERIMENTATEUR : Voici une réplique virtuelle de Domus…. présenter chacune des pièces, montrer les différents capteurs et les différents outils présentés lors de la visite. (Expliquer la différence entre la vision bureau et le salon)

Nous allons tester Domus Virtuel avec la phrase « Allumer une lampe quand je suis dans la salle de bain entre 20h et 22h » . On voit que cela marche aussi dans le Domus Virtuel et dans le Domus réel.

Debriefing pour lever les principaux problèmes IHM rencontrés avec l'application

DONNEES RECUEILLIES: VIDEO ET AUDIO

Notre application est en phase d'évaluation donc n'hésitez pas à nous faire part des points qui n'ont pas été clairs pour vous. Peut-être que je suis allé un peu vite et que je n'ai pas été très clair dans mes explications.

Avez-vous des questions à me poser concernant l'appartement ?
Voulez-vous le revisiter ?
Avez-vous des questions à me poser concernant le langage naturel ?
Voulez-vous refaire un test ?
Avez-vous des questions à me poser concernant le Domus virtuel ?
Voulez-vous refaire un test ?
Avez-vous des questions à me poser concernant l'horloge virtuelle ?
Voulez-vous refaire un test ?

Faire utiliser le langage naturel, 3D et Domus par le sujet

DONNEES RECUEILLIES : TRACES, VIDEO ET AUDIO - ANNOTATION DES COMPORTEMENTS

Nous allons maintenant vous demander de programmer l'habitat. Pour cela nous allons vous lire un scénario (donner le scénario papier au participant) que vous devrez programmer avec le langage

naturel et ensuite vérifier si Domus Virtuel vous a compris. Je resterai auprès de vous tout au long de l'expérience.

SCENARIO A LIRE

Vous habitez dans cette maison intelligente qui contient des objets programmables : machine à café, volets, lumière, capteurs de présence, pose-serviettes, etc. Grâce à ces nouveaux dispositifs, vous allez pouvoir programmer votre maison et en particulier l'enchaînement des tâches du matin. Sur la table de chevet, un bouton vous permet de lancer la routine du matin. Le matin, vous avez l'habitude de vous réveiller à 7h. Vous aimez vous faire réveiller par une musique douce et qu'une lumière tamisée s'allume. Si vous restez au lit trop longtemps, vous aimeriez que l'intensité de la lumière et de la musique augmente. Quand vous êtes décidé à vous lever, vous appuyez sur le bouton et vous vous levez. La lumière change d'intensité. Le volume sonore passe à un niveau acceptable. Une fois que vous êtes levé, vous souhaitez que les volets s'ouvrent. Ensuite, vous allez dans la salle de bain, la lumière s'allume dès que vous rentrez dans la pièce. L'eau de la douche est préchauffée et une lumière projetée sur la douche vous indique la température de l'eau. Quand vous sortez de la douche, et commencez à vous sécher, la cafetière se déclenche de sorte que le petit-déjeuner soit prêt.
Tout n'est pas forcément envisageable ?

Nous vous demandons de faire cette programmation et d'expliquer à haute voix ce que vous faites et pourquoi. Vous pouvez utiliser le langage naturel, le simulateur et l'horloge virtuelle. Quand vous pensez avoir fini, dites-le nous. Si vous n'y arrivez pas, ne vous inquiétez pas le problème vient certainement de DOMUS.

EXPERIMENTATEUR : LAISSER LE PARTICIPANT FAIRE SA PROGRAMMATION, PENDANT CE TEMPS RESTER AVEC LUI ET NOTER SES ACTIONS.

Tests et modifications

DONNEES RECUEILLIES : VIDEO ET AUDIO ET ANNOTATIONS DES MODIFICATIONS SOUHAITEES, TRACES DES MODIFICATIONS
Nous allons maintenant faire le bilan avec vous sur ce que vous avez réussi à faire et sur ce qui été plus difficile.

Globalement que pensez-vous de cette expérience que vous venez de faire ?
A votre avis est ce que DOMUS a compris ce que vous vouliez faire ? Pourquoi ?
Qu'est ce qui a été facile ?
Qu'est ce qui a été difficile ?

Maintenant, nous allons regarder ce qui se passe dans l'appartement et vous me direz ce qui correspond à ce que vous souhaitiez et si vous souhaitez faire des modifications

EXPERIMENTATEUR : FAIRE LE TEST AVEC LE PARTICIPANT NOTER CE QU'IL VOUDRAIT MODIFIER
J'ai noté ce que vous souhaitez modifier. Je vous propose de faire ces modifications sur l'application en langage naturel et de refaire des tests.

Débriefing

DONNEES RECUEILLIES : VIDEO ET AUDIO ET PRISE DE NOTE .

Nous allons maintenant vous posez quelques questions sur le langage naturel et l'application Domus Virtuel.

Quels sont les points forts du langage naturel ?
les points faibles ?
Qu'avez-vous pensez du vocabulaire utilisé ?
Qu'avez-vous pensez de la manière de rédiger les phrases ?
Qu'avez-vous pensez de lui indiquer des plages horaires ?
Quels sont les points forts de Domus virtuel ?
les points faibles ?
Que pensez-vous de l'utilisation de l'horloge virtuelle ?

Questionnaire final et SUS (usability scale Brooke)

DONNEES RECUEILLIES : AUDIO ET Questionnaire papier

Pour terminer nous allons vous demander de remplir ce questionnaire.
Avez-vous d'autres remarques ou questions à nous poser ?

12 Annexe 4 – Questionnaire

Questionnaire d'évaluation

Je me sentais très confiant en utilisant le système	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
J'avais besoin d'apprendre beaucoup de choses avant de pouvoir utiliser ce système	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion

Les prochaines questions concernent DOMUS VIRTUEL. Vous répondrez à chacune des propositions selon l'échelle d'accord.

J'aimerais utiliser Domus Virtuel fréquemment	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
J'ai trouvé Domus virtuel inutilement complexe	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
Je pensais que Domus virtuel était facile à utiliser	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
Je pense que j'aurais besoin d'une personne technique pour être capable d'utiliser Domus virtuel	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
J'ai trouvé les différentes fonctions de Domus virtuel ont été bien intégrées	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
Je pense qu'il y a trop d'incohérence dans Domus virtuel	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
J'imagine que la plupart des gens pourrait apprendre à utiliser Domus virtuel rapidement	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
J'ai trouvé le Domus virtuel très difficile à utiliser	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
Je me sentais très confiant en utilisant Domus virtuel	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion
J'avais besoin d'apprendre beaucoup de choses avant de pouvoir utiliser ce Domus virtuel	○ Tout à fait d'accord	○ Plutôt d'accord	○ Plutôt pas d'accord	○ Pas du tout d'accord	○ Sans opinion

Enfin, quelques questions vous concernant

Pouvez vous indiquer votre prénom ☐

Votre année de naissance ☐

Votre profession ☐

Par rapport aux nouvelles technologies , vous sentez plutôt fan ou plutôt opposant ? merci de vous situer sur l'échelle ci-dessous

Opposant — Fan

Par rapport à l'application sur l'échelle fan et opposant où vous situez vous ?

Opposant — Fan