THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Adrien Carteron**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Une approche événementielle pour le développement de services multi-métiers dédiés à l'assistance domiciliaire

---

Soutenue le 22 décembre 2017, devant le jury composé de :

Président du jury :	Francine Krief,	Professeur à Bordeaux INP
Directeur de thèse :	Charles Consel,	Professeur à Bordeaux INP
Co-Directeur de thèse :	Nic Volanschi,	Chargé de recherche Inria Bordeaux
Rapporteurs :	Frederic Weis,	Maître de Conférence HDR, Université de Rennes 1
	Philippe Lalanda,	Professeur à l'Université Joseph Fourier de Grenoble
Examinateurs :	Francine Krief,	Professeur à Bordeaux INP
	Nikolaos Georgantas,	Chargé de recherche Inria

– 2017 –

ADRIEN CARTERON

UNE APPROCHE ÉVÉNEMENTIELLE POUR LE DÉVELOPPEMENT DE SERVICES MULTI-MÉTIERS DÉDIÉS À L'ASSISTANCE DOMICILIAIRE

INRIA BORDEAUX SUD-OUEST, FRANCE

LaBRI
Unité Mixte de Recherche CNRS (UMR 5800)
351 cours de la Libération
33405 Talence Cedex
France

Équipe PHOENIX, INRIA Bordeaux Sud-Ouest
200 avenue de la Vieille Tour
33405 Talence Cedex
France

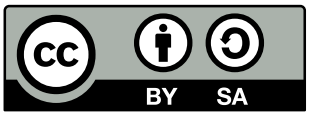
Université de Bordeaux

www.adrien-carteron.com

Inspired from the template provided by Paul van der Walt

WWW.DENKNERD.ORG

The typographic style of this document was inspired by Edward Tufte's book *Beautiful Evidence*, and typeset using LaTeX and a modified version of Kevin Godby's `tufte-book` class. The main text is typeset in *TEX Gyre Pagella*, which is based on Hermann Zapf's beautiful Palatino type face. The `typewriter` text is typeset in *Bera Mono*, originally developed by Bitstream, Inc.



Version : 5 avril 2019.

Résumé

La notion de contexte est fondamentale dans le champ de l'informatique ubiquitaire. En particulier lorsque des services assistent un utilisateur dans ses activités quotidiennes.

Parce qu'elle implique plusieurs disciplines, une maison équipée d'informatique ubiquitaire dédiée au maintien à domicile de personnes âgées demande l'implication d'une variété d'intervenants, tant pour concevoir et développer des services d'assistance, que pour déployer et maintenir l'infrastructure sous-jacente. Cette grande diversité d'intervenants correspond à une diversité de contextes. Ces différents contextes sont généralement étudiés séparément, empêchant toute synergie.

Cette thèse présente une méthodologie permettant d'unifier la conception et le développement de services sensibles au contexte et de répondre aux besoins de tout type d'intervenant.

Dans un premier temps, nous traitons les besoins des intervenants concernant l'infrastructure de capteurs/actionneurs : installation, maintenance et exploitation. Le modèle d'infrastructure de capteurs et un ensemble de règles en résultant permettent de superviser en continu l'infrastructure et de détecter des dysfonctionnements. Cette supervision simplifie le processus de développement d'applications, en faisant abstraction des problèmes d'infrastructure.

Dans un second temps, nous analysons un large éventail de services d'assistance domiciliaire dédié aux personnes âgées, en considérant la variété des besoins des intervenants. Grâce à cette analyse, nous généralisons l'approche de modèle d'infrastructure à tout type de services. Notre méthodologie permet de définir des services de façon unifiée, à travers un langage dédié, appelé Maloya, exprimant des règles manipulant les concepts d'état et d'évènement. Nous avons développé un compilateur de notre langage vers un langage événementiel dont l'exécution s'appuie sur un moteur de traitement d'évènements complexes (CEP).

Nous avons validé notre approche en définissant un large éventail de services d'assistance à la personne, à partir de services existants, et concernant l'ensemble des intervenants du domaine. Nous avons compilé et exécuté les services Maloya sur un moteur de traitement d'évènements complexes. Les performances obtenues en terme de latence et d'occupation mémoire sont satisfaisantes pour le domaine et compatible avec une exécution 24 heures sur 24 sur le long terme.

Mots clés : Assistance domiciliaire, Détection d'évènements complexes, Langage dédié, Sensibilité au contexte

Abstract

An event-driven approach to developing interdisciplinary services dedicated to aging in place

The notion of *context* is fundamental to the field of pervasive computing, and in particular when such services are dedicated to assist a user in his daily activities.

Being at the crossroad of various fields, a context-aware home dedicated to aging in place involves a variety of stakeholders to design and develop assistive services, as well as to deploy and maintain the underlying infrastructure. This considerable diversity of stakeholders raises correspondingly diverse context dimensions : each service relies on specific contexts (e.g., sensor status for a maintenance service, fridge usage for a meal activity recognition service). Typically, these contexts are considered separately, preventing any synergy.

This dissertation presents a methodology for unifying the design and development of various domestic context-aware services, which addresses the requirements of all the stakeholders.

In a first step, we handle the needs of stakeholders concerned by the sensors infrastructure : installers, maintainers and operators. We define an infrastructure model of a home and a set of rules to continuously monitor the sensor infrastructure and raise failure when appropriate. This continuous monitoring simplifies application development by abstracting it from infrastructure concerns.

In a second step, we analyze a range of services for aging in place, considering the whole diversity of stakeholders. Based on this analysis, we generalize the approach developed for the infrastructure to all assistive services. Our methodology allows to define unified services, in the form of rules processing events and states. To express such rules, we define a domain-specific design language, named Maloya. We developed a compiler from our langage using as a backend an event processing language, which is executed on a complex event processing (CEP) engine.

To validate our approach, we define a wide range of assistive services with our language, which reimplement existing deployed services belonging to all of the stakeholders. These Maloya services were deployed and successfully tested for their effectiveness in performing the specific tasks of the stakeholders. Latency and memory consumption performance turned out to be fully compatible with a 24/7 execution in the long run.

Keywords : Assistive services, Complex event processing, Domain-specific language, Context-aware

Remerciment

Ce manuscrit de thèse marque la fin d'une période riche de multiples enseignements. La lecture des manuscrits de mes prédécesseurs me fait prendre conscience qu'il est une évidence concernant la réalisation d'une thèse qu'il est nécessaire de rappeler ; c'est un travail collectif.

C'est pourquoi je tiens en premier lieu à remercier mes directeurs de thèse ; Charles Consel qui m'a fait confiance durant ces trois années en m'accueillant au sein de l'équipe Phoenix et en me permettant de bâtir un projet ambitieux. Nic Volanschi pour ses nombreuses suggestions, exigences de rigueur, et sa patience infinie.

Les rapporteurs Frederic Weis et Philippe Lalanda pour leurs relectures avisées méritent mes plus sincères remerciements. Merci aussi à Francine Krief qui à accepté de présider ma soutenance et Nikolaos Georgantas pour son écoute attentive et ses questions qui ont ouverts une autre dimension à mes travaux.

Je remercie tout spécialement Hélène Sauzéon qui, grâce à son expertise dans le vieillissement et les déficiences cognitives m'a permis d'appréhender les enjeux du domaine d'un point de vue pluridisciplinaire.

Je remercie également les membres de l'équipe Phoenix, présents, passés, perdus au combat, pour avoir participé à l'ambiance quotidienne, au support technique indispensable, pour les précieux conseils, l'accueil, bref pour ces quatre années inoubliables.

Je pense particulièrement aux anciens collègues, Charles F. des heures passées à refaire le monde autour de bières, Alex et ses conseils professionnels et sa folie incommensurable et Benjyx dont le rire hantera mes cauchemars durant des années.

Malgré la rigueur nécessaire à la réalisation de ces travaux, quelques pauses ont été marquées par leur association avec David Daney et ses conseils avisés, blagues, points de vue politique "objectifs".

Toutes ces personnes ont été autant d'apports constructifs et vents de fraîcheurs indispensables et ont participé d'une manière ou d'une autre à l'accomplissement de ce travail.

Enfin, et surtout je ne remercierai jamais assez Samantha, qui m'a soutenu pour que je donne le meilleur, même dans les moments les plus difficiles.

Table des matières

Table des figures

Liste des tableaux

1
Introduction

"Universal law is for lackeys. Context... is for kings." [1]

La notion de *contexte* est centrale à une variété de domaines en informatique, comme l'informatique mobile ou l'interaction homme-machine, mais tout particulièrement à l'informatique ubiquitaire [2] [3] [4]. Le contexte définit les circonstances dans lesquelles un système informatique est utilisé [5]. Il englobe beaucoup de dimensions [6] : connaître les interactions de l'utilisateur avec son environnement (*e.g.*, localisation géographique, ouverture de porte), mesurer ses signes physiologiques (*e.g.*, fréquence cardiaque), collecter des informations sur son environnement numérique (*e.g.*, email, agenda), superviser l'état des composants d'une infrastructure numérique (*e.g.*, matériels, logiciels, réseaux). Cette collecte d'informations est réalisée par des capteurs de tout type : des capteurs portés pour mesurer les activités physiologiques, par exemple ; des capteurs ambiants pour mesurer les interactions de l'utilisateur avec son environnement ; des capteurs logiciels pour mesurer des évènements numériques de l'utilisateur, comme un rendez-vous.

Les informations sur le contexte de l'utilisateur sont fondamentales pour l'informatique ubiquitaire puisqu'elles permettent à des services de s'adapter aux circonstances de leur utilisation. Ainsi, un service d'activités physiques adaptera ses recommandations à l'activité mesurée de l'utilisateur [7]. Un service d'économie d'énergie adaptera l'habitat aux présences et habitudes des occupants [8]. Un service de rappel d'activités du quotidien ne sollicitera l'utilisateur que si une activité d'intérêt n'est pas réalisée [9].

Le *domicile* a été l'objet d'une attention toute particulière de la part des chercheurs en informatique ubiquitaire [10] [11]. C'est un lieu central pour l'utilisateur qui met en œuvre tous les aspects de la notion de contexte : tous les types de capteurs sont pertinents ; un large éventail d'activités y sont réalisées ; et, une infinité de services peuvent être imaginés lorsque l'on prend en compte les spécificités des utilisateurs, leurs besoins

1. Gabriel Lorca : Captain of the U.S.S. Discovery (Star Trek Discovery Season 1 Episode 3)

2. Joëlle Coutaz, James L Crowley et al. [2005]. "Context is key". In : *Communications of the ACM* 48.3, p. 49–53
3. Anind K. Dey [2001]. "Understanding and Using Context". In : *Personal Ubiquitous Comput.* 5.1, p. 4–7.
4. Louise Barkhuus et Anind Dey [2003]. "Is Context-Aware Computing Taking Control away from the User ? Three Levels of Interactivity Examined". In : *UbiComp 2003 : Ubiquitous Computing : 5th International Conference, Seattle, WA, USA, October 12-15, 2003. Proceedings*. Springer Berlin Heidelberg, p. 149–156.
5. Coutaz, Crowley et al. 2005
6. Christine Bauer [2012]. "A Comparison and Validation of 13 Context Meta-Models." In : *ECIS*, p. 17.

7. Emil Jovanov et al. [2005]. "A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation". In : *Journal of NeuroEngineering and Rehabilitation* 2.1, p. 6.
8. M. Jahn et al. [2010]. "The Energy Aware Smart Home". In : *2010 5th International Conference on Future Information Technology*, p. 1–8.
9. L. Chen, J. Hoey et al. [2012]. "Sensor-Based Activity Recognition". In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6, p. 790–808.
10. Diane J Cook et al. [2013]. "CASAS : A smart home in a box". In : *Computer* 46.7, p. 62–69.
11. John Feminella et al. [2014]. "Piloteur : a lightweight platform for pilot studies of smart homes". In : *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM, p. 110–119.

et leurs préférences. La notion de contexte permet de déterminer quels services sont pertinents pour un utilisateur à un moment donné [12], et plus généralement, de développer des méthodes pour recueillir et analyser ces besoins [13].

Les enjeux du domicile équipé d'informatique ubiquitaire prennent une dimension sociétale lorsque l'on considère le maintien à domicile des personnes âgées. Dans ce domaine, le domicile doit fournir des services d'assistance pour pallier les pertes dues au vieillissement (*e.g.*, cognitives) et soutenir une vie indépendante [14]. Ces services couvrent deux domaines principaux : (1) superviser les activités du quotidien (*e.g.*, préparation de repas, toilette, se coucher) pour maintenir le statut fonctionnel de l'utilisateur [15] et (2) détecter les situations potentiellement dangereuses (*e.g.*, cuisinière, porte d'entrée) pour garantir la sécurité de l'utilisateur [16].

Dans le domaine du maintien à domicile des personnes âgées, une maison équipée d'informatique ubiquitaire repose sur des expertises appartenant à différentes disciplines. Au-delà des personnes âgées elles-mêmes, on trouve les aidants professionnels ou informels, les experts en vieillissement, les professionnels de santé, les développeurs d'applications ainsi que les techniciens de maintenance. Cette grande diversité d'intervenants se reflète par une grande diversité de besoins en matière d'informations contextuelles. Les besoins de services de chaque intervenant reposent sur des contextes spécifiques (état des capteurs pour la maintenance, utilisation du frigidaire pour l'assistance à domicile. etc.). Ces différents contextes sont généralement étudiés séparément, en silo. Typiquement, chaque intervenant développe sa propre approche, pour extraire ses informations contextuelles, empêchant toute synergie. En outre, cette extraction d'informations est généralement programmée avec des couches logicielles génériques (librairies, middleware) qui ne sont pas nécessairement adaptées à cette collecte d'informations et qui ne favorisent pas la réutilisation et la factorisation d'une expertise.

Par conséquent, couvrir l'ensemble des besoins de services pour le maintien à domicile de personnes âgées demande de déployer différentes solutions, dédiées aux différentes problématiques du domaine, pour couvrir les différents intervenants. Cette situation ne permet pas le passage à l'échelle dans un environnement écologique, car une hétérogénéité dans les solutions à déployer complique le déploiement, la maintenance, ainsi que la création de nouveaux services. L'expression de besoins de services de chaque intervenant dans le domaine de l'assistance domiciliaire pourrait être unifiée par la mutualisa-

12. AJ Brush et al. [2011]. "Home automation in the wild : challenges and opportunities". In : *proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, p. 2115–2124.

13. Joëlle Coutaz, Emeric Fontaine et al. [2010]. "DisQo : A user needs analysis method for smart home". In : *Proceedings of the 6th Nordic Conference on Human-Computer Interaction : Extending Boundaries*. ACM, p. 615–618

14. Parisa Rashidi et Alex Mihailidis [2013]. "A survey on ambient-assisted living tools for older adults". In : *IEEE journal of biomedical and health informatics* 17.3, p. 579–590.

15. Loïc Caroux et al. [2014]. "Verification of Daily Activities of Older Adults : A Simple, Non-Intrusive, Low-Cost Approach". In : *ASSETS - The 16th International ACM SIGACCESS Conference on Computers and Accessibility*. Rochester, NY, United States, p. 43–50.

16. Rashidi et Mihailidis 2013

tion des informations de contexte.

Contributions

Nous proposons une approche unifiée de définition de contextes couvrant la diversité des préoccupations des intervenants d'une maison équipée d'informatique ubiquitaire. En particulier, nous étudions notre approche dans le domaine du maintien à domicile et faisons ainsi levier sur une étude expérimentale incluant plus de cent personnes âgées, équipées d'une plate-forme d'assistance domiciliaire. Cette étude nous permet de réaliser une analyse de besoins s'appuyant sur des cas d'usage pratiques, émanant d'un éventail d'expertises, comme par exemple : l'assistance à la réalisation des activités du quotidien (*e.g.,* préparation de repas), la sécurisation du domicile (*e.g.,* alerter l'utilisateur lorsqu'une porte d'entrée est ouverte et non surveillée pendant un certain temps), ou bien la maintenance de l'infrastructure (*e.g.,* alerter l'exploitant en cas d'absence de communication entre le capteur et la passerelle).

Notre approche unifiée repose sur un paradigme événementiel dédié à la détection de contextes.

Langage dédié

Notre première contribution porte sur la conception d'un langage dédié à la définition de services dans un domicile sensible au contexte. Ce langage, nommé Maloya, fournit des constructions de haut niveau pour exprimer des services sensibles au contexte avec (1) les concepts relatifs au domaine de l'assistance domiciliaire et (2) des opérateurs de composition permettant de manipuler ces concepts. En outre, Maloya permet de couvrir les besoins de services des intervenants impliqués dans domicile sensible au contexte. Enfin, notre langage fournit à la fois un cadre conceptuel et des outils pour concevoir et développer des services domiciliaires pour les personnes âgées.

Implantation

Nous avons développé un compilateur de notre langage vers un langage d'évènements. La compilation de Maloya comprend plusieurs étapes pour implanter les abstractions contextuelle du langage, tout en masquant la complexité propre aux traitements d'évènements temporels. L'exécution des règles compilées en langage d'évènements s'appuie sur un moteur existant de traitement d'évènements complexes (CEP) et une architec-

ture logicielle *centrée données*, unifiant les sources de données hétérogènes. Des flux d'évènements, produits sous forme canonique, alimentent le moteur CEP qui exécute les règles. Ces règles peuvent être modifiées pendant l'exécution.

Validation

Nous avons validé notre approche écrivant dans notre langage 55 services d'assistance existants et couvrant le spectre des besoins des intervenants du maintien à domicile. Les règles écrites dans notre langage sont compilées dans un langage événementiel et déployées pendant une période de temps suffisante pour évaluer la performance des services en conditions réelles d'utilisation.

Modèle d'infrastructure

Une déclinaison de notre approche, particulièrement importante du point de vue pratique, est la supervision en continu d'une infrastructure de capteurs. Celle-ci consiste à factoriser les préoccupations de fiabilité de capteurs entre les différents services contextuels déployés sur une plateforme, sous la forme d'un modèle d'infrastructure. Ce concept peut être formulé complètement dans notre paradigme évènementiel sous la forme d'un ensemble de services de surveillance. Le modèle d'infrastructure permet dès l'étape de déploiement de vérifier le bon positionnement des capteurs, pour fournir ainsi des informations contextuelles fiables aux différents services, et d'assurer par la suite le bon fonctionnement des capteurs, libérant ainsi les autres services de cette préoccupation.

Organisation du manuscrit

Ce document est organisé comme suit.

Le **Chapitre 2** présente un état de l'art de la sensibilité au contexte dans l'informatique ubiquitaire. Nous discutons des différentes méthodologies d'implantation de services sensibles au contexte.

Le **Chapitre 3** présente une première brique de notre approche adressant les besoins de fiabilité d'une infrastructure de capteurs et actionneurs. Elle montre que la définition d'un modèle d'infrastructure permet 1) de s'assurer du bon positionnement des capteurs, dès la phase de déploiement, 2) de superviser en continu l'infrastructure et 3) de faciliter le déve-

loppement de services.

Le **Chapitre 4** procède à une analyse des besoins contextuels dans le domaine de l'assistance domiciliaire de personnes âgées. À travers l'étude de différents scénarios couvrant la variété de besoins d'intervenants, nous isolons les concepts et besoins communs des services d'assistance domiciliaire.

Le **Chapitre 5** présente le coeur de notre approche, constituée d'un langage de règles, nommé Maloya, et d'une architecture logicielle sous-jacente. Maloya permet d'exprimer aisément un ensemble de services d'assistance domiciliaire en restreignant son expressivité aux seuls concepts du domaine. Les règles sont exécutées sur un flux continu d'évènements de contexte.

Le **Chapitre 6** décrit plus en détail les étapes de compilation de notre langage vers un langage événementiel. Nous validons le langage Maloya et son compilateur par une expérimentation en grandeur réelle dans une plate-forme d'assistance domiciliaire

Enfin, le **Chapitre 7** conclut notre étude et détaille des pistes pour de futurs travaux.

2
État de l'Art

La recherche en informatique ubiquitaire est indissociablement liée à la notion de contexte. Il existe de nombreuses approches pour capturer le contexte et définir des services sensibles au contexte. Dans ce chapitre, nous examinons le domaine de l'implantation de services sensibles au contexte pour le maintien à domicile.

Sommaire

Aperçu

— Une analyse des approches de la capture du contexte.

— Une analyse des méthodes pour concevoir des services sensibles au contexte.

— Une analyse des enjeux de la sensibilité au contexte dans le maintien à domicile des personnes âgées.

Informatique Ubiquitaire

La notion d'informatique ubiquitaire a été décrite pour la première fois par Weiser [Weiser 1993] dans une définition que nous traduisons comme suit.

> L'objectif est de réaliser le type de technologie le plus efficace, celui qui est essentiellement invisible pour l'utilisateur. Pour amener les ordinateurs à ce point tout en conservant leur puissance, il faudra des types d'ordinateurs radicalement nouveaux, de toutes tailles et formes, à la disposition de chaque personne. J'appelle ce monde futur "l'informatique ubiquitaire"

Cette notion d'informatique ubiquitaire est définie par la composition d'éléments matériels et logiciels ; il doit être invisible pour les utilisateurs dans leur quotidien. Dans cette relation entre l'utilisateur et l'ordinateur, ce dernier doit être capable d'assurer toutes ses fonctionnalités tout en restant à l'écart. Pour compenser le manque d'interactions explicites avec l'utilisateur, il est nécessaire de prendre en compte des informations implicites, regroupées sous le terme de *contexte*.

La notion de contexte

Depuis cette première définition de l'informatique ubiquitaire, les progrès technologiques (miniaturisation, fiabilité de capteurs), ont permis de préciser la vision de Mark Weiser. Les ordinateurs peuvent être à la fois physiquement cachés à l'utilisateur et les capteurs permettent aux ordinateurs d'interagir avec l'utilisateur sans son intervention directe. Dès lors, le contexte prend une dimension centrale dans l'informatique ubiquitaire et sa définition évolue, comme le montre celle proposée par Anind Dey [A. K. Dey 2001].

> Le contexte correspond à n'importe quelle information qui peut être utilisée pour caractériser l'état d'une entité. Une entité est une personne, un lieu, ou un objet. Cette entité est considérée comme pertinente pour l'interaction entre un utilisateur et une application, incluant l'utilisateur et l'application eux-mêmes.

Cependant, Coutaz *et al.* apportent une nuance à la définition de contexte en expliquant que *"le contexte n'est pas seulement l'état d'un environnement prédéfini par un ensemble fixe de ressources d'interaction. Il fait partie d'un processus d'interaction avec un environnement en constante évolution composé de ressources reconfigurables, migratoires, distribuées et multi-échelles"* [Coutaz, Crowley et al. 2005].

Cette vision présente le contexte comme une situation qui se produit et peut évoluer, il ne peut donc pas être caractérisé comme une entité fixe. Son exploitation est alors soumise à

des techniques de machine-learning. Bien que différentes, ces deux visions du contextes peuvent être complémentaires, notamment dans le domaine de l'assistance domiciliaire, où les informations contextuelles peuvent varier au cours du temps en fonction de l'évolution des capacités cognitives de l'utilisateur. Ainsi, la définition proposée par Anind Dey, comme source d'information d'une situation qui se produit, permet aux concepteurs de services de définir des services utilisant explicitement le contexte comme source d'information, pour prendre des décisions d'une manière déterministe, selon les besoins précis d'un utilisateur à un moment donné. La notion de contexte évolutif peut être utilisée pour concevoir des services moins déterministes, qui s'adaptent par exemple à l'évolution de ses besoins dans le temps. Néanmoins, le déterminisme d'une application d'assistance peut être un critère influant sur son acceptabilité par des personnes âgées, qui sont souvent fortement routinisées [Caroux et al. 2014].

Sensibilité au contexte

Pour être sensible au contexte, un système doit *"utiliser le contexte pour fournir des informations pertinentes et/ou des services à l'utilisateur, où la pertinence dépend de la tâche de l'utilisateur"* [A. K. Dey 2001]. Par exemple si à la fin d'une période définie de petit déjeuner, les informations de contexte relatif à la préparation de celui-ci n'ont pas été détectées, alors le système peut envoyer un message à l'utilisateur pour lui rappeler de prendre son petit déjeuner. En d'autres termes, un système est dit "sensible au contexte" si il est capable d'utiliser le contexte [Alegre et al. 2016].

Pour être accepté par l'utilisateur, tout en étant intégré dans son environnement, Barkhuus et Dey proposent trois principes d'interaction d'un système sensible au contexte [Barkhuus et A. Dey 2003] ; (1) personnalisation : l'utilisateur doit pouvoir définir manuellement ses préférences et attentes ; (2) sensibilité passive au contexte : le système surveille continuellement l'environnement et offre le choix à l'utilisateur d'agir ; (3) sensibilité active au contexte : le système surveille continuellement l'environnement et agit automatiquement. De ces trois principes, nous retenons l'importance de pouvoir spécifier facilement et finement des services de détection de contextes. En effet, l'existence même de tels services sous-tend à la fois la sensibilité passive et active. De plus, leurs précision et facilité de développement est une condition cruciale pour la personnalisation des services contextuels. L 'approche que nous présenterons par la suite tâchera donc de répondre à ces besoins,

à travers un langage dédié de haut niveau et suffisamment expressif.

Dans une analyse de commonalités sur 13 méta modèles de contexte formalisés dans la littérature, et environ 300 publications sur les applications d'informatique ubiquitaire, Bauer a identifié six niveaux de *dimensions contextuelles* largement utilisées dans l'informatique ubiquitaire [Bauer 2012] : le monde physique, le monde logiciel, l'utilisateur, ses activités, le contexte social et leurs dynamiques. Pour être sensible au contexte, un système doit être capable de capturer ces dimensions contextuelles et de les traiter pour fournir un service. Nous retenons ici la nécessité d'une large expressivité, couvrant toutes ces dimensions, pour une approche visant le développement de services contextuels.

Perera *et al.* [Perera et al. 2014] ont classé le cycle de vie de l'information de contexte en quatre séquences ; (1) l'acquisition, qui définit la façon dont la donnée est acquise, selon cinq facteurs (*e.g.,* fréquence de l'évènement, source, type de capteur, *etc.*) ; (2) la modélisation, qui correspond à la technique choisie pour la modélisation du contexte (*e.g.,* graphique, orientée objet, ontologie, *etc.*) ; (3) le raisonnement, qui exprime la façon dont va être créé un nouveau "savoir" à partir des données de contexte (*e.g.,* apprentissage, règles, probabilités, *etc.*) ; et (4) la diffusion, qui est la manière dont l'information obtenue va être livrée à son consommateur (*e.g.,* requête ou abonnement). Le modèle d'infrastructure de capteurs que nous introduirons, constitué de règles de surveillance, tâchera de répondre aux points (2) et (3) ci-dessus, pour le cas particulier où les informations de contexte concernent la fiabilité d'une maison connectée.

Acquisition du contexte

L'acquisition du contexte repose sur la collecte d'informations de mesure d'environnement à partir de capteurs de tous types. Pour fournir un service, un système ubiquitaire doit être capable de composer avec des capteurs hétérogènes. Pour ce faire il existe des approches fournissant une abstraction sur le réseau de capteurs, et d'autres définissant une sémantiques aux capteurs aussi uniforme que possible, malgré leur hétérogénéité. Par ailleurs, le placement des capteurs étant un élément central pour la mesure d'un contexte, d'autres approches se concentrent sur la modélisation du placement des capteurs.

Réseaux de capteurs

Un réseau de capteurs est un système composé de différents appareils aux capacités de calcul plus ou moins limitées, capable de mesurer l'environnement et éventuellement d'agir sur ce dernier. Ces capteurs communiquent le plus souvent par ondes et sont alimentés par batteries. Beaucoup de recherches visent à simplifier la programmation d'applications destinées à l'infrastructure de capteurs, en définissant des modèles de programmation adaptés. Sugihara et Gupta [Sugihara et Gupta 2008] présentent un recensement complet de ces approches. Les auteurs définissent une taxonomie de modèles de programmation en fonction de leur niveau d'abstraction. Ils classent les approches de programmation de réseaux de capteurs en modèles de bas niveau et haut niveau.

Modèles de bas niveau Ces modèles de programmation permettent de s'abstraire des préoccupations matérielles de chaque nœud de capteur pour en simplifier la programmation. Par exemple, le système d'exploitation TinyOs [Hill et al. 2000] fournit un support de programmation via nesC, un langage dédié au domaine des réseaux de capteurs [Gay et al. 2003] dérivé du C. Greenstein *et al.* [Greenstein et al. 2004], avec SNACK fournissent aux développeurs un langage de composition et une bibliothèque. Cet ensemble permet le développement de bibliothèques de services à combiner directement dans les applications.

Modèles de haut niveau Ces modèles offrent un point de vue centré sur l'application pour programmer les réseaux de capteurs. Les approches de haut niveau se concentrent sur la façon dont les nœuds collaborent pour partager, agréger et traiter les données mesurées. Ces approches peuvent être organisées en fonction du niveau auquel les nœuds collaborent (*i.e.*, groupe, réseau) [Sugihara et Gupta 2008]. Ces approches offrent la possibilité de programmer le réseau de capteurs (1) en utilisant des langages de requêtes [Madden et al. 2003], similaires à celles utilisées dans les bases de données, pour interroger les données de capteurs, (2) écrire des programmes fonctionnels pour masquer la manipulation des états des nœuds [Whitehouse, Sharp et al. 2004], ou (3) étendre les langages de programmation existants pour prendre en compte la nature distribuée des capteurs [Gummadi et al. 2005].

Ces approches simplifient la programmation de la couche applicative sur une infrastructure de réseau de capteurs, en proposant des abstractions à différents niveaux de ce réseau.

Ces abstractions fournissent des supports efficaces pour orchestrer un système distribué hétérogène, en aidant dans la répartition des calculs, la communication et la synchronisation entre les nœuds, et en cachant les différences entre des capteurs physiques. Dans le cas d'un domicile connecté, ces aspects sont habituellement gérés par une boîte ("Box") domotique. Par ailleurs, les travaux concernant les réseaux de capteurs fournissent peu de support pour décrire la logique applicative. Notamment, il manque des abstractions de haut niveau, telles que des relations temporelles entre les évènements et états des différents capteurs. Ainsi, implanter des conditions complexes impliquant plusieurs capteurs (e.g. le frigidaire est ouvert pendant que le four est allumé, le tout pendant le créneau du petit déjeuner) peut s'avérer difficile en l'absence d'un support adéquat.

Sémantique des capteurs

D'autres approches permettent d'abstraire, dans les applications sensibles au contexte utilisant des capteurs, les détails concernant les capteurs, comme la gestion de mesures bas niveau ou le traitement des dysfonctionnements. La sémantique de flux, proposée par Whitehouse *et al.* [Whitehouse, Zhao et al. 2006] permet aux applications d'effectuer des requêtes sémantiques, telles que "détecter un véhicule", plutôt que d'interroger directement un magnétomètre bas niveau et de vérifier si sa valeur excède un certain seuil. L'implantation est basée sur Prolog et permet de composer des capteurs et des unités d'inférence. Issus du génie logiciel, les capteurs logiciels offrent une abstraction sur chaque capteur matériel. Dans la proposition de Lin [Lin 2004] ces capteurs logiciels décrivent des services implantés dans un middleware sur Jini, une architecture distribuée basée sur Java. Cette approche permet de combiner de multiples capteurs de manière flexible. Il existe également des standards exposant les capteurs en tant que services Web, comme le standard Sensor Web Enablement (SWE), porté par le consortium OGC [Botts et al. 2008]. Ce standard permet d'améliorer l'interopérabilité des services sur de nombreux aspects (représentation des mesures, découverte des capteurs, flux de données des capteurs, *etc.*). Au-dessus du standard SWE, La Semantic Sensor Web [Sheth et al. 2008] ajoute une sémantique de méta données, exploitant des ontologies et des règles d'inférences (également standardisée par la W3C). Par exemple, une règle d'inférence déclarative, exprimée dans le Semantic Web Rule Language peut définir un capteur pour les conditions de tempête, construit à partir de capteurs de

température, de vent, et de précipitation.

Nous retenons de ces travaux l'importance que peut jouer une couche d'abstractions sémantique pour faciliter le développement de services contextuels. En conséquence, nous reprendrons la vue des mesures fournis pas les capteurs comme un flux d'évènements, le plus uniforme possible. Il reste à trouver un niveau adéquat d'abstraction pour une maison connectée. Par exemple, il est évident que le développement de services sera plus simple si on considère des évènements du type "cafetière utilisée pendant 3 minutes" plutôt que la séquence correspondante des consommations électriques.

Placement de capteurs

Des efforts de recherche ont été déployés pour systématiser et vérifier le placement de capteurs. Hong *et al.* [Hong et al. 2013] proposent une méthodologie empirique de décomposition pour suivre automatiquement le placement de capteurs dans les pièces d'un immeuble et déterminer si les capteurs peuvent être déplacés et si la configuration physique du bâtiment peut évoluer. À l'échelle plus modeste d'une maison connectée, la dynamique des capteurs ne semble pas un problème important, mais certains enseignements de ce travail peuvent être utiles lorsqu'un service doit s'exécuter en même temps dans une grande variété de configurations domiciliaires.

Murao *et al.* [Murao et al. 2013] ont pour but de déterminer le meilleur positionnement de capteurs portés pour surveiller une activité. Ils proposent des fonctions d'évaluation qui permettent d'évaluer la position d'un capteur en terme de précision de reconnaissance et de portabilité du capteur. À cause des contraintes qu'ils imposent, de nombreux utilisateurs refusent les capteurs portés pour surveiller les activités du quotidien. Surie *et al.* [Surie et al. 2008] valident la capacité de reconnaissance d'activités d'un réseau de capteurs en comparant les résultats d'identification de l'activité avec la capture des images de cette activité par une caméra portée par l'utilisateur. Même si cette approche fournit des informations sur la fiabilité de l'infrastructure, le temps nécessaire à l'analyse vidéo empêche tout passage à l'échelle. De plus, les caméras sont perçues comme étant trop intrusives pour être utilisées dans les domiciles des personnes, particulièrement à des endroits comme une chambre ou une salle de bain. L'approche présentée par Philipose *et al.* [Philipose et al. 2004] pour la reconnaissance d'activités consiste à placer des tags RFID sur les objets du quotidien qui sont pertinents pour l'activité. La reconnaissance d'interaction se fait donc par l'utilisation de

gants intégrant un lecteur RFID. Cette approche évite les problèmes de positionnement de capteurs, mais peut introduire des faux positifs causés par la portée d'émission du gant RFID et la distance entre les objets tagués.

À la différence de ces approches implémentatoires, Beckmann *et al.* [Beckmann et al. 2004] définissent cinq principes de conception pour guider l'installation de capteurs dans un domicile.

— Utiliser une technologie familière, de façon appropriée par rapport à la vision conceptuelle qu'en a l'utilisateur. Par exemple il est plus aisé pour un utilisateur de concevoir qu'une ouverture de porte est détectée en plaçant un capteur de contact, plutôt que d'utiliser une caméra.

— Prendre en compte l'usage du domicile dans les choix d'implantation de capteurs. Le placement de capteurs dans un environnement soulève des questions à la fois esthétiques et des questions sur la durabilité de leur positionnement sur le long terme en fonction de l'environnement (*e.g.*, présence d'animaux, d'enfants *etc.*).

— Éviter l'utilisation de capteurs trop sensibles à la direction de leur placement. En effet, des capteurs comme des caméras et des microphones, mis à part les questions de vie privée qu'ils peuvent soulever, sont des capteurs dont l'efficacité de la mesure est extrêmement sensible à la façon dont ils sont orientés. Ces capteurs demandent donc une grande précision dans la façon dont ils sont placés, ce qui augmente les risques d'un mauvais positionnement à l'installation.

— Détecter la mauvaise installation de composants et fournir un résultat même pour une installation partielle. Le résultat de services sensibles au contexte dépend des mesures d'interaction ; il est donc indispensable de pouvoir détecter une installation de capteurs incorrecte, pour la rectifier et pouvoir fournir ces services. De plus, des problèmes d'installation (défaillante ou incomplète) ne doivent pas empêcher les services de fonctionner, même si cela peut impliquer un mode dégradé.

— Informer l'utilisateur sur la façon dont les données sont collectées, stockées et transmises. Cette étape permet à l'utilisateur de comprendre comment ses interactions avec son environnement sont mesurées. Cette phase d'éducation permet de démystifier le système ubiquitaire, facilitant son appropriation par l'utilisateur.

Nous considérons ces principes comme très pertinents pour faciliter l'acceptation et pour assurer le bon fonctionnement

d'une plateforme d'assistance domiciliaire. Cependant, ce ne sont là que des principes. Pour les mettre en œuvre, il reste à fournir un support concret pendant le cycle de vie des applications d'assistance, tel qu'un modèle explicite de l'infrastructure, et des services de surveillance continuelle de ce modèle pendant l'exécution des applications déployées.

Beaucoup de travaux de recherche qui se concentrent sur des plates-formes de domiciles connectés visent à impliquer l'utilisateur final dans les étapes d'installation et/ou de maintenance de l'infrastructure de capteurs. En particulier, Kawsar *et al.* [KAWSAR et al. 2008] proposent un cadre de développement pour supporter le déploiement de capteurs par un utilisateur final dans un domicile ubiquitaire. Combiner ce cadre de développement avec les cinq principes de Beckmann *et al.* [BECKMANN et al. 2004] facilite le positionnement des capteurs et leur configuration individuelle par l'utilisateur. Cependant, aucun outil n'est fourni pour contrôler le bon positionnement effectif des capteurs et vérifier leur fiabilité. Sadoun *et al.* [SADOUN et al. 2011] partent d'une description en langage naturel de l'installation et des scénarios pour en déduire un modèle d'environnement. Ce modèle est une ontologie OWL accompagnée de règles d'inférence en SWRL. Utilisant des règles d'inférence automatique, la cohérence du modèle et la conformité des scénarios décrits avec le modèle peuvent être vérifiées. Cette vérification peut signaler à la fois les scénarios incohérents et les incohérences dans la description de l'installation. Cependant, cette approche ne vérifie que le placement de l'installation (notamment les problèmes de couvertures), et non son fonctionnement écologique. En particulier, leurs règles d'inférences ne raisonnent pas en terme d'intervalles de temps, ce qui est une notion clé pour beaucoup d'applications d'assistance.

Définition de services

Un service sensible au contexte traite différentes informations de contexte pour fournir une information de plus haut niveau. Il existe plusieurs types d'approches permettant d'effectuer ce traitement :

— De type middleware ;

— Basées sur des automates ;

— Par traitement d'évènements.

Mis à part ces approches de programmation de services, destinées aux programmeurs professionnels, plusieurs travaux pro-

posent des techniques permettant à l'utilisateur final de définir lui-même des services.

Approches de type middleware

Certaines approches pour développer des applications sensibles au contexte reposent sur un middleware pour fournir aux programmeurs, utilisant un langage de programmation général, des abstractions pour opérer et gérer des équipements (*e.g.*, découverte de capteurs) et des services sensibles au contexte. Avec FedNet, Kawsar *et al.* [Kawsar et al. 2008] proposent un framework permettant aux applications d'être exprimées comme des collections de tâches indépendantes de l'implantation. HomeOS, de Rosen *et al.* [Rosen et al. 2004], permet de définir des interfaces pour s'abstraire d'appareils connectés spécifiques. Román *et al.* [Román et al. 2002] avec Gaia et Ballesteros *et al.* [Ballesteros et al. 2006] avec Plan B, proposent des approches basées sur des systèmes de fichiers, en prolongeant la philosophie des systèmes Unix au delà des périphériques habituels d'entrée/sortie, pour couvrir aussi les capteurs/actionneurs. Ranganathan *et al.* [Ranganathan et al. 2005] avec Olympus, présentent une approche séparant les préoccupations relatives à la gestion d'un environnement ubiquitaire dynamique. Ainsi, un administrateur a la possibilité de définir des règles Prolog pour gérer la politique relative à l'environnement. Le développeur, quant à lui, peut définir ses services sans se soucier de certaines contraintes de l'environnement.

Augmentant le niveau d'abstraction, d'autres approches introduisent un processus de développement déclaratif rigoureux, dédié aux services sensibles au contexte. Des exemples d'approches de développement outillées sont DiaSuite [Bertran et al. 2014 ; Cassou et al. 2012] et IoTSuite [Patel et Cassou 2015].
Ces approches automatisent certaines parties routinières de la tâche de programmation en générant un cadre de programmation spécialisé pour un environnement donné, décrit par une hiérarchie de capteurs/actionneurs avec leurs interfaces de programmation respectives.

Toutes ces approches orientées middleware ont pour but de fournir une abstraction sur la gestion des détails de l'infrastructure de capteurs. En cela, elle fournissent une bonne base pour le développement de services contextuels mais ne supporter pas les besoins de programmation applicative, à savoir la reconnaissance des interactions avec l'environnement et de leurs combinaisons.

Approches orientées automates

Plusieurs types d'approches exploitent des modèles d'automates et leurs outils associés. En effet, les contextes peuvent être représentés par des séquences particulières d'évènements dans l'environnement, contraintes dans leur ordre et dans leurs temporisation. Par exemple, la situation de porte non surveillée peut être identifiée par un automate acceptant un évènement de porte ouverte et un évènement de porte fermée, séparés par un délai temporel supérieur à une certaine valeur. De tels modèles d'exécution de reconnaissance de contextes peuvent être exprimés avec des langages dédiés pour modéliser les automates. Ces langages peuvent être visuels (StateCharts, SyncCharts) ou textuels (langages synchrones) [Gamatié 2010]. Les délais temporels, dans ces modèles, doivent être gérés explicitement en utilisant des minuteries externes pour générer des évènements d'expiration de délais. Les automates temporels [Bengtsson et Yi 2004] ajoutent l'expression native des délais temporels dans le modèle. Ces modèles à base d'automates sont habituellement accompagnés par des outils formels pour prouver des propriétés temporelles utiles au modèle, tels que l'atteignabilité d'un état. Ils sont assez expressifs pour manipuler tous les cas requis par les services d'un domicile sensible au contexte, mais imposent aux développeurs d'implanter des motifs communs comme le séquençage d'évènements, la reconnaissance d'un ensemble d'événements au cours d'un état, *etc.* Lorsqu'ils implantent ces motifs de base sous forme d'automates temporels, les développeurs peuvent introduire de subtiles erreurs ou de légères variations dans le comportement qui devrait être identifié. Ce problème est amplifié quand les modèles sont écrits par les intervenants avec différents niveaux d'expertise.

Approches de traitement événementiel

D'autres approches conceptualisent également les informations de capteurs comme un flux de données d'évènements temporels, dans lequel des séquences peuvent être identifiées [Krishnan et Diane J. Cook 2014]. Ces approches cherchent à offrir des modèles plus haut niveau que les automates pour décrire des séquences d'évènements d'intérêt. C'est le cas du langage IFTTT [Augusto et Chris D. Nugent 2004], qui intégre des opérateurs temporels sur des événements d'environnement.

Plus généralement, il existe un domaine qui, historiquement, n'est pas lié à l'informatique ubiquitaire et offre un paradigme

de traitement de flux d'évènements : le traitement d'événements complexes [Cugola et Margara 2012] (CEP comme *Complex Event Processing*). Ce domaine introduit des opérateurs de composition qui implantent des motifs d'événements fréquents, comme des séquences ordonnées dans un laps de temps, des évènements alternatifs, *etc.* Certains CEP associent à un évènement complexe, l'intervalle de temps comprenant tous les évènements y contribuant, et utilisent l'algèbre d'intervalles d'Allen [Allen 1983] pour combiner les évènements entre eux [Anicic et al. 2010 ; Li et al. 2011 ; Helmer et Persia 2016 ; Hausmann 2014]. Cette approche permet de prendre en considération des évènements non ponctuels, ayant une certaine durée. Toutefois, cette sémantique d'intervalle ne fournit pas pour autant une notion d'état associée à des objets physiques courant d'un domicile connecté. Par exemple, l'état d'une porte qui est ouverte doit être implantée avec un évènement complexe commençant par l'ouverture d'une porte et finissant par la fermeture d'une porte, sans autre évènement de porte entre les deux. Ce type d'encodage est sujet aux erreurs et a tendance à complexifier les formules CEP. Néanmoins, les CEP peuvent constituer une excellente base pour une approche introduisant les bonnes abstractions du domaine de l'assistance domiciliaire, tout en assurant leur encodage uniforme et prévisible par des formules CEP.

Programmation par les utilisateurs

Ces approches partent des besoins de l'utilisateur final du domicile ubiquitaire, et lui fournissent des langages dédiés pour développer des applications sensibles au contexte par lui-même. Ces langages dédiés sont habituellement complétés par un environnement de développement dédié. Ces approches de programmation dédiées à l'utilisateur final ont abouti à des langages textuels et visuels. Scratch, présenté par Resnick *et al.* [Resnick et al. 2009], offre des notations de programmation visuelle, couvrant la plupart des usages d'un langage de programmation général. En principe, ce langage permet à l'utilisateur d'écrire une variété de programmes dans le domaine des domiciles ubiquitaires mais cette approche n'autorise pas des abstractions spécifiques au domaine de l'assistance domiciliaire. IFTTT (If This Then That)[1] offre des notations graphiques plus spécialisées pour automatiser des processus simples impliquant des services web, des capteurs et des actionneurs. Cette approche réduit l'effort conceptuel de l'utilisateur final au détriment d'une réduction drastique de l'expressivité du langage. Par exemple, les conditions permettent l'utilisation

1. https://ifttt.com

d'un unique capteur ou service, et ne peuvent pas être composées. De plus, l'absence de distinction possible entre les états et les évènements, peut rendre leur utilisation confuse [HUANG et CAKMAK 2015]. Améliorant IFTTT, Coutaz et Crowley [COUTAZ et CROWLEY 2016], présentent AppsGate qui contient un langage textuel dédié. Ce langage permet de faire une distinction claire entre les états et les évènements, et permet quelques compositions limitées des conditions dans les règles. AppsGate propose également un environnement de développement pour l'utilisateur final dédié aux domiciles ubiquitaires. Il a été validé par des expériences avec des utilisateurs portant sur des règles simples d'automatisation dédiées au confort. Même si cette extension d'IFTTT est prometteuse, elle n'est cependant pas satisfaisante pour couvrir les besoins d'applications d'assistance domiciliaire. Par exemple, les auteurs expliquent qu'"*exprimer des conditions composées [impliquant de multiples états ou évènements] est difficile*" [COUTAZ et CROWLEY 2016]. De plus, même de simples compositions temporelles telles que "A précède immédiatement B" ne peuvent pas être exprimées dans ce langage, dû à l'absence de connecteurs booléens dans les conditions.

La plate-forme Casensa [CRIEL et al. 2011], utilise une ontologie, appelée CAEMP, qui permet à l'utilisateur d'instancier des entités, dans le sens que l'utilisateur crée une représentation virtuelle d'une entité physique en définissant diverses propriétés. À partir de ces entités il peut leur attribuer des *comportements intelligents* à partir d'un catalogue disponible en ligne. Cependant, cette approche reste centrée sur les développeurs pour ce qui est de la création de ces comportements intelligents.

Autonomie domiciliaire des personnes agées

Veillir chez soi est à la fois une solution permettant de diminuer le coût pour la société d'une population vieillissante, et un confort pour la personne. Cependant, plusieurs facteurs peuvent compromettre le maintien à domicile d'une personne agée : d'une part, les pertes d'autonomie dues au vieillissement ; d'autre part, le fardeau pour l'entourage ou les aidants professionnels avec les risques psychosociaux associés. Dans ce contexte, un domicile équipé d'informatique ubiquitaire sensible au contexte peut favoriser le maintien à domicile des personnes âgées et réduire le fardeau des aidants.

Il existe de nombreuses plates-formes d'assistance domiciliaire dédiées au vieillissement à domicile qui ont été déve-

loppées ces dernières années [Chan et al. 2008 ; Rashidi et Mihailidis 2013]. Ces plates-formes répondent à des besoins d'assistance que l'on peut classer en trois catégories [Nehmer et al. 2006] :

— Les traitements d'urgences, relatifs aux situations de danger pour l'utilisateur. Il s'agit de détecter les situations anormales [Hoque et al. 2015].

— Le renfort de l'autonomie. Il s'agit de tous les services qui vont compenser la perte d'autonomie de l'utilisateur. Le renfort d'autonomie passe souvent par l'observation des activités du quotidien [Lee et A. K. Dey 2015].

— Le confort. Ces services sont relatifs au lien social [Baecker et al. 2014].

Cependant, la plupart des approches sont dédiées à la résolution d'un problème donné parmi ce spectre de besoins. Or une assistance domiciliaire efficace requiert que l'ensemble des problèmes soit couvert. Pour une utilisation in-vivo, il est difficile de concevoir le déploiement d'une technologie par problème à adresser, ne serait ce que pour en assurer la maintenance.

Une autre limitation quasi-générale des travaux de recherche existants est que la plupart de ces systèmes sont validés uniquement par des expériences en environnement contrôlé : les participants se rendent dans un appartement de recherche pour effectuer des tâches spécifiques [Rashidi et Mihailidis 2013 ; Rantz et al. 2011]. Les conclusions de ces études sont ainsi difficile à généraliser pour prédire le fonctionnement de la plateforme et de ses services dans un environnement naturel à long terme.

Synthèse

Le développement de systèmes sensibles au contexte suit quatre étapes explicitées par Alegre *et al.* [Alegre et al. 2016].

1. Le **recueil des besoins** dépend du domaine dans lequel s'intègre le système sensible au contexte. S'il est admis que l'utilisateur final doit être impliqué dans la conception des services, l'ensemble des intervenants gravitant autour du système n'est pas encore pris en compte dans la conception [Alegre et al. 2016]. L'intégration de l'ensemble des intervenants est critique dans le domaine du maintien à domicile. Par exemple les aidants naturels et professionnels sont des sources d'informations indispensables sur l'assistance à fournir à l'utilisateur, et les techniciens de maintenance sont indispensables à la fiabilité de l'infrastructure.

2. L'**analyse et la conception** se font selon les besoins recueillis. Cette étape inclut principalement la définition d'une architecture logicielle et/ou le choix de modèles de conception (design patterns) adaptés au développement de services contextuels. La littérature recense déjà des modèles de conceptions adaptés aux services orchestrant des capteurs et actionneurs, tels que le modèle SCC, utilisé avec succès dans le passé dans plusieurs domaines, y compris l'assistance domiciliaire. Notons que ce modèle est même supporté par des outils de développement [Bertran et al. 2014]. Le choix d'une architecture est plus spécifique au domaine cible, mais il existe des principes architecturaux comme les architectures centrées données [Qin et al. 2016] ou les architectures dirigées par les données [Chen, C. D. Nugent et al. 2012], qui semblent bien adaptés à une maison connectée.

3. L'**implantation** de services est largement couverte par la littérature. De nombreuses approches proposent même des méthodes destinées aux utilisateurs finaux. Cependant, ces approches sont soit (1) trop générales par rapport au domaine de l'informatique ubiquitaire, et n'offrent pas les abstractions ou l'expressivité nécessaires ; soit (2) trop spécifiques pour couvrir le spectre des besoins d'assistance du domaine ciblé.
   Cette étape inclut aussi le choix des capteurs pour mesurer l'environnement, et le choix de la plate-forme.

4. Le **déploiement** est une étape clé. En effet, c'est à cette étape que les capteurs sont installés ; il faut donc s'assurer de leur bon positionnement. Or il n'existe à l'heure actuelle que peu de travaux couvrant ce besoin. La **maintenance** est également une étape importante. Si certaines plates-formes intègrent des mécanismes de qualité de services, peu d'approches intègrent directement la possibilité d'exprimer des services de maintenance de l'infrastructure. Cela s'explique par le fait qu'il existe encore peu d'expérimentations écologiques à large échelle et la fiabilité de l'infrastructure mise en place sur le long terme n'y est donc pas encore primordiale.

La spécification de services sensibles au contexte doit pouvoir être proche du domaine auquel ses services sont appliqués, pour qu'ils correspondent aux besoins des intervenants. En outre, les services doivent pouvoir utiliser des contextes exprimés de façon suffisamment large pour prendre en compte tous les besoins de contextes. En particulier, cette expressivité doit permettre également d'exprimer un modèle d'infrastructure, où les informations de contexte concernent la fiabilité

continue d'une maison connectée.

Dans un domaine comme l'assistance domiciliaire des personnes âgées, où l'acceptabilité de la technologie est parfois critique, il est indispensable que tous les intervenants soient impliqués, et qu'ils puissent s'approprier au mieux la technologie en leur permettant d'exprimer eux même des services grâce à leur culture du domaine de l'assistance et en limitant le besoin de culture informatique dans cette conception.

3
Rendre le domicile sensible au contexte

Ce chapitre présente une approche visant à assurer la fiabilité de la sensibilité au contexte pour les applications d'assistance domiciliaire. Cette approche consiste à rendre explicite un *modèle de rôles* joué par les capteurs, à travers des règles, contribuant à assurer que le fonctionnement du capteur correspond à son rôle. Ces *règles de conformités* permettent de fournir des instructions détaillées quant à l'installation et le positionnement de capteurs dans le monde physique et assurer la conformité d'une installation déployée par rapport à son modèle durant son exploitation [1].

Sommaire

Contributions

— Une approche qui améliore la fiabilité des applications sensibles au contexte pour la surveillance d'activité.

— Un concept de *modèle de rôles de capteurs* est introduit pour capturer les besoins des applications en terme de capteurs, et permet de vérifier en continu l'infrastructure de capteurs en parallèle de l'exécution des applications.

— Ce modèle de rôles de capteurs peut être *factorisé* entre les applications, *partagé* entre les intervenants, et *utilisé* pour les évolutions de la plate-forme.

1. Ce travail à fait l'objet d'une publication : A. Carteron et al. [2016]. "Improving the Reliability of Pervasive Computing Applications by Continuous Checking of Sensor Readings". In : *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, p. 41–50.

Dans un domicile équipé d'informatique ubiquitaire, la sensibilité au contexte permet à une application de notifier l'utilisateur à propos d'une activité[2], *uniquement* si l'activité a été oubliée ; elle permet de déclencher une alerte pour une porte restée ouverte *uniquement* si il n'y a personne dans la zone ; elle permet d'appeler l'utilisateur lorsque la cuisinière est restée allumée, *uniquement* si elle n'est pas surveillée depuis un certain temps. La sensibilité au contexte devient alors une clé essentielle pour que ces applications puissent être intégrées à la vie quotidienne de l'utilisateur. En effet, des notifications issues de reconnaissances de contexte erronées fatiguent l'utilisateur et ralentissent, voire empêchent l'acceptation de la technologie. Il est donc essentiel de minimiser les faux positifs dans la détection de situations nécessitant l'attention de l'utilisateur.

2. Marie Chan et al. [2008]. "A review of smart homes—Present state and future challenges". In : *Computer Methods and Programs in Biomedicine* 91.1, p. 55–81.

La fiabilité d'une application ubiquitaire et sensible au contexte est définie par la fiabilité du système dans son ensemble[3][4]. En informatique ubiquitaire, le système est principalement constitué d'une couche logicielle et d'une couche matérielle. La fiabilité logicielle a été amplement étudiée ; des approches à base de simulation ont été proposées[5] pour tester la fiabilité du logiciel avant son déploiement. Ces approches permettent de tester, in-vitro, des scénarios sur des applications, en simulant les interactions de l'utilisateur avec son environnement (*e.g.*, une ouverture de porte, une présence dans une pièce, la mise en marche d'un appareil). Malgré la rigueur et l'exhaustivité que permet la mise en œuvre de cette phase de test, la fiabilité logicielle reste dépendante de la fiabilité de la couche matérielle elle-même, fournissant les données de contexte.

3. J.A. Stankovic et al. [2005]. "Opportunities and obligations for physical computing systems". In : *Computer* 38.11, p. 23–31.

4. Sarah Mennicken et al. [2014]. "From Today's Augmented Houses to Tomorrow's Smart Homes : New Directions for Home Automation Research". In : *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington : ACM, p. 105–115.

5. J. Bruneau et al. [2009]. "DiaSim : A parameterized simulator for pervasive computing applications". In : *Mobile and Ubiquitous Systems : Networking Services, MobiQuitous, 2009. MobiQuitous '09. 6th Annual International*, p. 1–10.

La fiabilité de la couche matérielle d'un système dépend de nombreux facteurs. D'un point de vue unitaire, la fiabilité de la détection d'un contexte relève de la fiabilité du capteur sous-jacent. Elle peut être assurée simplement en vérifiant que le capteur fonctionne correctement. Cependant, pour être sensible au contexte, une application peut avoir besoin de combiner des données en provenance de plusieurs capteurs[6]. Conceptuellement, un capteur est vu comme jouant un *rôle* dans la logique applicative : il mesure les interactions des utilisateurs avec leur environnement. Ainsi, le capteur est supposé être placé de façon appropriée (orientation, fréquence d'échantillonnage, absence d'interférences, *etc.*) pour mesurer efficacement ces interactions[7]. Quand la caractérisation d'un contexte dépend d'une combinaison d'interactions, le développeur construit un modèle implicite de rôles joués par une combinaison de capteurs[8]. Par exemple, un capteur de contact sur la porte d'entrée peut être couplé avec un capteur de mouvement couvrant la zone de l'entrée. Une application peut utiliser

6. Stankovic et al. 2005

7. W.Keith Edwards et RebeccaE. Grinter [2001]. "At Home with Ubiquitous Computing : Seven Challenges". English. In : *Ubicomp 2001 : Ubiquitous Computing*. Sous la dir. de GregoryD. Abowd et al. T. 2201. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 256–272.

8. Karen Henricksen et al. [2002]. "Modeling Context Information in Pervasive Computing Systems". English. In : *Pervasive Computing*. Sous la dir. de Friedemann Mattern et Mahmoud Naghshineh. T. 2414. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 167–180.

cette paire de capteurs comme suit : si la porte d'entrée est restée ouverte pendant un certain temps, mais que la présence de l'utilisateur est détectée à proximité pendant tout ce temps, alors aucune alerte ne doit être envoyée. Cependant, il y a certaines dépendances entre les capteurs couplés, dues au fait qu'une porte ne s'ouvre jamais toute seule. Ces dépendances font partie du modèle implicite construit par le développeur. Si ce modèle implicite est transgressé par une défaillance du capteur de mouvement, l'application sensible au contexte est compromise. Il peut en résulter une notification erronée alertant d'une situation de porte d'entrée ouverte et non surveillée, et ce, même si l'utilisateur est a proximité.

Pour adresser la fiabilité des capteurs, le développeur introduit généralement du code pour tester la validité de ce modèle implicite. Cette stratégie consiste à retranscrire des *règles implicites* via des instructions conditionnelles réparties à travers le code de l'application. Ces règles contribuent à assurer que la lecture des capteurs est conforme au modèle implicite. Par exemple, si une cuisine est équipée d'un capteur de contact sur un certain placard dans la cuisine et d'un capteur de mouvement dans la cuisine, alors, la porte du placard ne doit pas être détectée comme étant ouverte sans la détection préalable d'une présence dans la cuisine. Si cette situation se produit, alors il y a transgression du modèle implicite. De nombreuses raisons peuvent être la cause de cette violation (placement, dysfonctionnement) et doivent conduire à une intervention pour résoudre le problème. Détecter ces situations est un élément essentiel de la fiabilité des applications d'informatique ubiquitaires sensibles au contexte déployées dans un domicile [9].

9. Chris Beckmann et al. [2004]. "Some Assembly Required : Supporting End-User Sensor Installation in Domestic Ubiquitous Computing Environments". English. In : *UbiComp 2004 : Ubiquitous Computing*. Sous la dir. de Nigel Davies et al. T. 3205. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 107–124.

Étude de cas

Pour illustrer notre approche, nous présentons, comme étude de cas, la surveillance de deux activités de cuisine. Ces exemples sont simples mais complets. Ils utilisent différents types de capteurs pour reconnaître différents types d'interactions entre l'utilisateur et son environnement dans la cuisine.

La surveillance d'activité est extrêmement dépendante de la population ciblée par les applications d'assistance [10]. Par exemple, une personne âgée peut simplement avoir besoin d'un rappel pour la réalisation de ses activités quotidiennes [11], alors qu'une population avec déficience intellectuelle (*e.g.*, syndrome de Down) peut avoir besoin de supervision pour l'exécution d'étapes clés d'une tâche [12].

Pour déterminer *quelles* activités doivent être surveillées et

10. Jeannette Durick et al. [2013]. "Dispelling Ageing Myths in Technology Design". In : *Proceedings of the 25th Australian Computer-Human Interaction Conference : Augmentation, Application, Innovation, Collaboration*. OzCHI '13. Adelaide, Australia : ACM, p. 467–476.

11. Caroux et al. 2014

12. Dany Lussier-Desrochers et al. [2016]. "Analysis of How People with Intellectual Disabilities Organize Information Using Computerized Guidance". In : *Disability and Rehabilitation : Assistive Technology*. To appear.

comment elles doivent être surveillées, nous avons besoin d'expertise sur la population ciblée. Des professionnels comme des ergonomes et des ergothérapeutes fournissent cette expertise. Dans notre exemple, un expert définit les activités qui doivent être surveillées dans la cuisine. À partir de cet ensemble d'activités, l'expert détermine *quelles interactions* avec l'environnement doivent être détectées pour identifier ces activités. Typiquement, l'expert demande à l'utilisateur de simuler les différentes étapes effectuées pour réaliser son activité [13]. Une fois que les interactions clés concernant chaque activité d'intérêt ont été identifiées, nous devons déterminer *quels capteurs* sont pertinents pour mesurer ces interactions. Cette étape a été étudiée par Beckmann *et al.* [Beckmann et al. 2004]. Ils présentent un guide pratique pour installer les capteurs et l'évaluent à travers une étude utilisateurs.

13. Caroux et al. 2014

Supposons maintenant que l'expert définit les activités suivantes : (1) la surveillance de la préparation du petit déjeuner, pour envoyer un rappel à l'utilisateur si cette activité a été manquée et (2) la surveillance de la cuisinière pour s'assurer de son utilisation en toute sécurité.

Surveillance de la préparation du petit déjeuner

Pour surveiller la préparation du petit déjeuner, l'expert du domaine définit des scénarios typiques réalisés par l'utilisateur. Un de ces scénarios peut par exemple être décrit ainsi : l'utilisateur (1) fait du café en utilisant une machine à café électrique, (2) prend une tasse depuis un placard spécifique ou le lave-vaisselle, et (3) prend du lait dans son frigidaire. À partir de ce scénario, il est possible d'extraire les interactions à mesurer : (1) allumer la machine à café, (2) ouvrir la porte du placard, (3) ouvrir la porte du frigidaire. Pour mesurer ces interactions, un capteur de consommation électrique est associé à la machine à café : si une consommation électrique est captée, alors la machine à café est allumée. Les interactions avec la porte du frigidaire sont détectées au moyen d'un capteur de contact. Pour simplifier nous n'avons pas placé de capteur de contact sur le lave-vaisselle, et supposons que dans ce cas, l'action de prendre une tasse ne peut pas être détectée.

Sécurité de l'utilisation de la cuisinière

Le signalement des problèmes de sécurité liés à la cuisinière dépend principalement de la détection de son utilisation et de la présence d'un utilisateur à proximité pour surveiller la cuisson. Un capteur de consommation électrique est utilisé pour

détecter si la cuisinière est en fonctionnement, et un capteur de mouvement est positionné pour détecter une présence dans un périmètre stratégique autour de la cuisinière (typiquement l'ensemble la cuisine).

Rendre explicite un modèle implicite

Pour résumer, les activités à surveiller dans notre cas d'utilisation sollicitent la reconnaissance de cinq interactions avec l'environnement de la cuisine : ouvrir la porte d'un placard spécifique, ouvrir la porte du frigidaire, allumer la machine à café, allumer la cuisinière, et détecter la présence dans la cuisine.

Pour schématiser, dans le contexte de notre cas d'utilisation, la dernière étape de notre méthode consiste à rendre explicite le modèle implicite qui assure la fiabilité des interactions détectées. Nous savons qu'un capteur de mouvement couvre l'ensemble de la cuisine, donc, chacune des interactions mesurées dans la cuisine doit être précédée par la détection d'une présence. Ainsi, si une interaction est mesurée sans être précédée par la détection d'une présence dans la cuisine, il est probable que les capteurs impliqués ont un dysfonctionnement. Ces règles contribuent à assurer qu'une infrastructure de capteurs fonctionne en conformité avec le modèle de rôles.

Besoins des applications

Les applications de support d'activité domiciliaire fournissent des services de surveillance et d'assistance pour un utilisateur. Ces services reposent sur un ensemble de données contextuelles relatives à des mesures d'interactions de l'utilisateur avec son environnement, comme illustré avec notre cas d'étude.

Cependant, il y a souvent un fossé entre les données brutes, fournies par les capteurs, et le point de vue conceptuel du développeur d'applications sur les interactions avec l'environnement. Par exemple, un capteur de contact produit des valeurs booléennes définissant les états ouvert/fermé. Cet état doit être combiné avec l'emplacement spécifique (et/ou un identifiant) du capteur pour rendre l'interaction exploitable par l'application. De plus, plusieurs capteurs combinés peuvent être requis pour détecter une interaction. Par exemple, la détection d'une présence dans une pièce en forme de L nécessite plusieurs capteurs de mouvements.

Rôle

Du point de vue de l'application, le rôle définit les informations d'interaction directement exploitables par la logique applicative. Ce rôle consiste en un type d'interaction (*e.g.*, présence) et l'emplacement de l'interaction (*e.g.*, la cuisine). D'un point de vue physique, un rôle définit les besoins qui doivent être satisfaits par un ou plusieurs capteurs pour détecter une certaine interaction. Conceptuellement, les rôles sont positionnés entre la couche physique des capteurs et les applications, comme représenté en Figure 1. Pour être en conformité avec un rôle, les capteurs doivent être disposés correctement en terme de position et d'orientation. Comme illustré précédemment, détecter une présence dans une cuisine en forme de L nécessite au moins deux détecteurs de mouvements, orientés de façon à couvrir entièrement la cuisine, sans pour autant détecter les mouvements des espaces adjacents (*e.g.*, le couloir adjacent à la cuisine). Il est à noter que les rôles permettent la séparation des préoccupations entre les applications et la couche physique des capteurs.

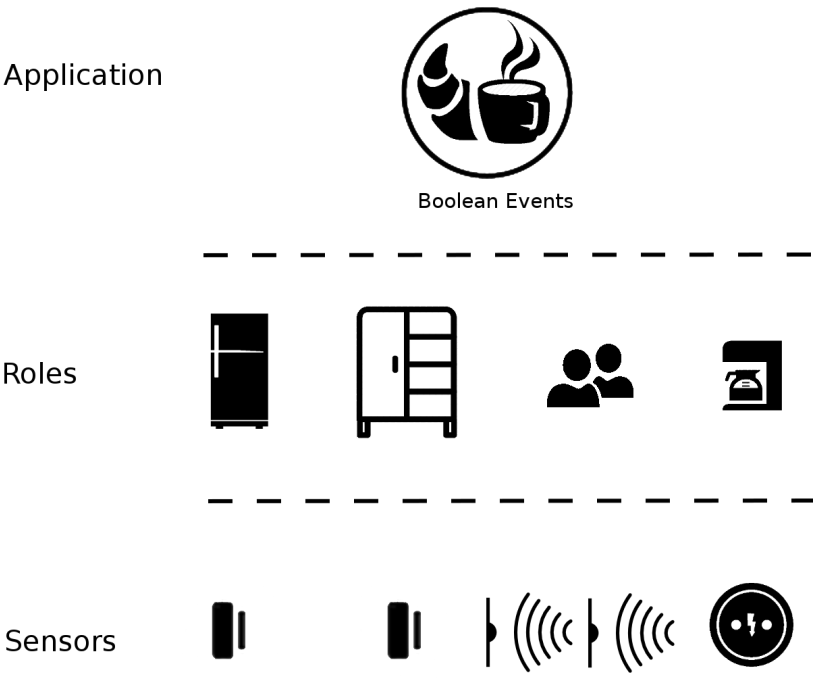


FIGURE 1: Besoins d'application et rôles.

Sémantique des rôles

La sémantique d'un rôle couvre (1) la détection d'une interaction avec l'environnement, produisant une valeur *true*, et (2) la détection de la fin de cette interaction, produisant une valeur

false. Ce comportement est illustré dans la Figure 2. Un rôle est alors une fonction définie sur le temps et couvrant des valeurs booléennes. Quand une interaction est détectée à un instant, celle-ci produit true, jusqu'à ce qu'elle ne soit plus détectée, alors elle produit false.

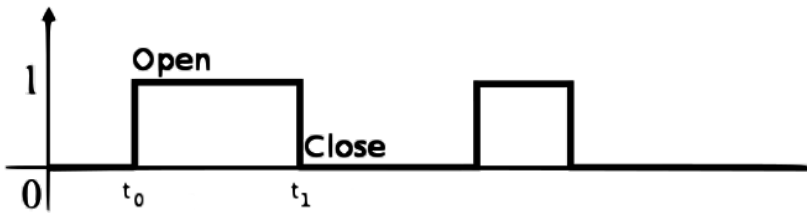


FIGURE 2: Sémantique des rôles.

En pratique, pour fournir cette sémantique temporelle à l'application, l'implémentation d'un rôle doit être filtrée pour isoler les séquences de données atypiques (bruit). Par exemple, pour un rôle joué par un capteur de contact, s'il fournit deux valeurs true (ouverture) consécutives, l'implementation filtrera habituellement la seconde valeur.

Modèle d'infrastructure

Les capteurs installés dans un domicile forment une infrastructure qui supporte les rôles requis par les applications déployées. La bonne implantation des rôles est critique pour la fiabilité de la sensibilité au contexte de cette infrastructure et donc des applications qui reposent sur cette infrastructure. Cette fiabilité va au-delà des tests unitaires de chaque rôle.

Pour adresser la fiabilité de l'infrastructure, nous proposons de construire un modèle de cette infrastructure qui peut être vérifié. Ce modèle ne prend pas uniquement en compte les rôles individuels, mais également leur conformité par des règles globales portant sur l'infrastructure de capteurs.

Les évènements de rôle

La première étape pour expliciter le modèle d'infrastructure par un ensemble de règles, est de délimiter le domaine des objets sur lesquels les règles pourront agir : les évènements de rôle. Un évènement de rôle se compose de trois éléments : (1) une interaction qui s'est produite (2) à une localisation donnée, (3) pendant une période de temps spécifique. Tout d'abord examinons la notion de période. Elle est définie en tant qu'inter-

valle délimité par deux horodatages. Ainsi :

$$Period = \mathbb{N}^2$$
$$\forall\ p \in Period, p =< t_1, t_2 >\ and\ t_1\ <\ t_2$$

Une période peut également être vue comme un ensemble de valeurs de temps croissantes (ou d'instants), allant de t_1 à t_2, séparées chaque fois d'une seconde – une granularité plus fine n'est pas nécessaire en pratique. Nous pouvons alors utiliser les opérations habituelles sur les ensembles pour opérer avec les périodes, comme $\subseteq, \supseteq$.

Enfin, un évènement de rôle est une interaction qui est arrivée pendant une période. L'ensemble d'interactions est définie par *Inter* (*e.g.,* Présence, Ouverture, Utilisation). Un ensemble de localisations, *Loc*, spécifie les emplacements d'intérêt dans le domicile (*e.g.,* Cuisine, Salle de bain, Chambre). Les évènements de rôle sont donc définis comme suit.

$$e\ \in\ Event = Inter \times Loc \times Period$$

Lorsque l'infrastructure de capteurs surveille le domicile, elle produit des logs de données structurés en flux d'évènements de rôles, définis précédemment. Le log d'évènements de rôles est défini ainsi $log\ \in\ Log = \mathscr{P}(Event)$

Formuler des règles

Maintenant que les logs d'évènements de rôles sont définis et peuvent ainsi être manipulés, nous nous intéressons aux règles du modèle d'infrastructure. Ces règles sont exprimées par un ensemble de formules logiques dans le calcul de prédicats du premier ordre. Nous introduisons ces règles en examinant trois exemples de notre cas d'utilisation.

Présence dans la cuisine. Cette règle rend explicite la dépendance des capteurs dans la cuisine. En substance, nous voulons exprimer le fait que chaque interaction détectée, qui n'est pas un mouvement, doit être encadrée par une interaction de mouvement. Ce faisant, nous exprimons le fait que le capteur de mouvement dans la cuisine couvre toutes les autres interactions dans la cuisine (*e.g.,* porte de placard, machine à café). Une fois exprimée, cette sémantique assure la conformité des relevés de capteurs dans la cuisine.

Notre règle de présence dans la cuisine est définie ainsi :

$$\forall\ < i, Kitchen, p > \in\ Log, i \neq Presence\ \Rightarrow$$
$$\exists\ < Presence, Kitchen, p' > \in\ Log,\ p \subseteq p'$$

Portes restées ouvertes. Nous supposons que dans un but d'assistance domiciliaire, une porte équipée d'un capteur de contact

ne doit pas rester ouverte au-delà d'une certaine durée, notée MAX. Une telle règle s'applique typiquement sur la porte du frigidaire et sur la porte d'entrée parce que dans des conditions d'utilisation normales d'une maison, elles ne peuvent pas rester ouvertes très longtemps. La durée MAX peut varier en fonction des préférences de l'utilisateur et du type de porte.

Cette règle est définie ainsi :

$$\forall \ < Opening, l, p > \in \ Log \Rightarrow \#p < MAX$$

Notons qu'une porte restée ouverte très longtemps peut être associée soit à un dysfonctionnement du capteur (comme pour les portes suscitées, de l'entrée ou du frigidaire), soit à un oubli de l'utilisateur de fermer la porte si celle-ci n'est pas surveillée par une application déclenchant une notification de sécurité. Par exemple, la porte du placard de notre cas d'utilisation est surveillée uniquement pour rappeler à l'utilisateur de préparer ses repas, et non pas pour lui rappeler qu'elle est restée ouverte.

Non omniprésence. Certaines règles de conformité peuvent être spécifiques à un certain domaine d'application. Par exemple, nos recherches en informatique ubiquitaire sont principalement concentrées sur l'autonomie domiciliaire de personnes âgées vivant seules. Cette situation nous permet de définir la règle de conformité suivante : un rôle de présence ne peut pas être détecté simultanément à deux emplacements différents. La règle de non omniprésence est définie ainsi.

$$\forall \ < Presence, l, p > \in \ Log \Rightarrow$$
$$\nexists \ < Presence, l', p' > \in \ Log, \ l \neq l' \wedge \ p' \cap p \neq \varnothing$$

En pratique, définir des règles de conformité doit être associé à la fourniture d'instructions précises pour installer et positionner les capteurs dans le monde physique. Par exemple, la règle de présence dans la cuisine nécessite que la présence soit reconnue dans toute la cuisine. Une fois un domicile installé, les règles assurent la conformité entre l'installation et le modèle.

Validation

Pour valider le concept de modèle d'infrastructure à base de règles, nous présentons tout d'abord une architecture pour vérifier en continu qu'une installation est conforme à son modèle. Ensuite, nous décrivons une implantation de cette architecture ; celle-ci nous a permis de valider l'approche expérimentalement.

Architecture

Globalement, l'architecture que nous proposons consiste à abstraire les lectures brutes de capteurs à travers une couche de rôles, alimentant à la fois les applications avec des valeurs de plus haut niveau, et les logs d'évènements de rôles utilisés par les règles de conformité du modèle d'infrastructure. Cette architecture est décrite en Figure 3.

Les évènements de rôles sont traités simultanément par les applications et le module de vérification du modèle. Ce faisant, les règles de conformité peuvent être exécutées à la volée pour relever les erreurs quand elles apparaissent. Alternativement, les règles peuvent être exécutées hors ligne, sur des log déjà enregistrés, pour diagnostiquer des problèmes lorsque un opérateur est disponible pour de la maintenance.

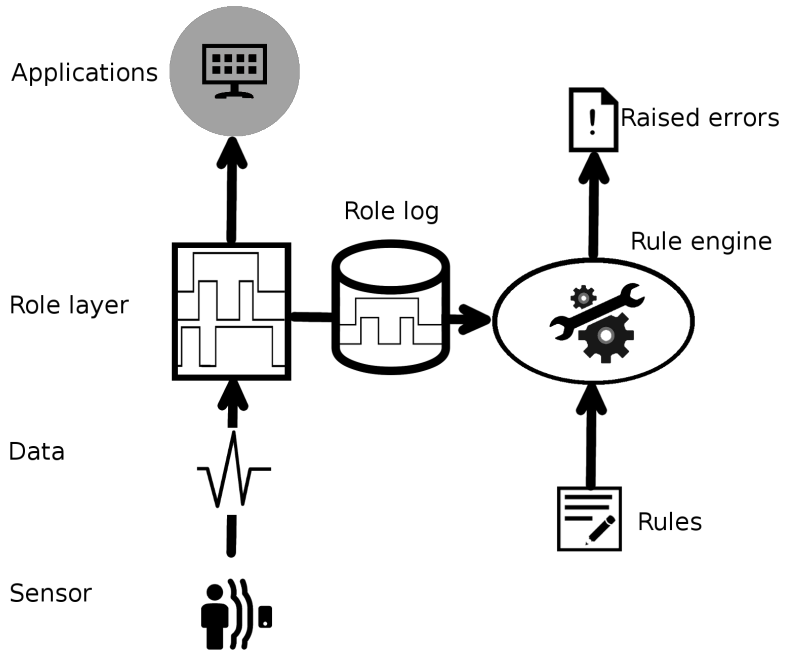


FIGURE 3: Architecture de notre approche de vérification continue d'une infrastructure par rapport à un modèle.

Au-delà de la maintenance, les logs d'évènements de rôle peuvent aussi être précieux pour analyser l'évolution des activités quotidiennes des personnes âgées. En effet, ces logs permettent des analyses longitudinales qui peuvent, le cas échéant, révéler des dégradations du fonctionnement quotidien dues au vieillissement. Ces analyses permettent au professionnel d'adapter l'assistance en supprimant ou installant de nouvelles applications pour satisfaire l'évolution des besoins des utilisateurs.

En outre, en faisant levier sur les logs d'évènements de rôle, la reconnaissance d'activités du quotidien peut être ajustée.

Par exemple, le seuil de déclenchement des notifications peut être modifié pour prévenir la fatigue de l'utilisateur. De plus, ces logs peuvent servir pour rejouer des séquences d'interactions et déboguer des applications au comportement erroné.

Règles. Les règles de conformité sont implantées sous la forme de prédicats Prolog, utilisant des opérateurs manipulant les évènements de rôles.

Analyseur de rôles. La plate-forme d'assistance utilisée pour valider notre approche ne fournissant pas la couche d'abstraction nécessaire pour produire les rôle, nous simulons cette couche dans notre implantation en utilisant un "analyseur de rôles". Ce composant construit un log d'évènements de rôles en transformant le log de lecture de capteurs et les informations associées (type de capteur, localisation, état et horodatage). Cet analyseur de rôle est implanté par un module C++.

Moteur de règles. Ce module parcourt le log d'évènements de rôle produit par l'analyseur de rôles, et appelle un interpréteur Prolog pour exécuter chaque règle. De cette façon, quand une règle échoue, le moteur de règles identifie les évènements de rôle impliqués et produit une liste d'évènements de rôles non conformes avec les règles qu'ils ont fait échouer. Ce module est également implanté en C++.

Le moteur de règles et l'analyseur de rôles totalisent plus de 2700 lignes de codes C++, pour transformer les données brutes de capteurs en évènements de rôles, et alimenter les règles en Prolog.

L'expérimentation DomAssist

Le projet DomAssist [14] vise à prolonger l'autonomie des personnes âgées dans leur propre domicile, en leur fournissant une plate-forme d'assistance domiciliaire avec des applications d'aide à la réalisation des activités quotidiennes. Des ergothérapeutes, psychologues et experts en vieillissement ont défini les activités à surveiller et l'ensemble des interactions à mesurer avec l'environnement.

14. http://phoenix.inria.fr/research-projects/homeassist

Pour ce projet, deux types d'applications ont été définies : (1) des applications pour surveiller les activités du quotidien et pour assister l'utilisateur quand elles n'ont pas été effectuées, et (2) des applications pour sécuriser le domicile (*e.g.*, porte d'entrée restée ouverte). Une première étude utilisateur a été conduite en recrutant 24 participants et en déployant à leur domicile notre plate-forme d'assistance domiciliaire pour une durée de neuf mois. Nous avons utilisé les données collectées

durant cette première étude du projet DomAssist pour valider notre modèle d'infrastructure.

Modèle

Les capteurs utilisés dans cette expérimentation permettent de mesurer douze interactions différentes avec l'environnement. Les interactions de présence sont mesurées avec des capteurs de mouvement, les ouvertures sont mesurées avec des capteurs de contact, et l'utilisation d'appareils électriques est mesurée avec des capteurs de consommation électrique. La configuration de DomAssist en terme de capteurs et leur rôles associés est résumée dans la Table 1.

Table 1: Rôles dans DomAssist.

Room	Role	Sensor
Kitchen	Coffee maker in use	EM
	Cabinet door open	CS
	Fridge door open	CS
	Microwave in use	EM
	Presence	CS
Entrance	Door open	CS
	Presence	MD
Bathroom	Shower in use	MD
	Presence	MD
Bedroom	Dressing open	CS
	Bedside lamp in use	EM
	Presence	MD

EM = Electric Meter, CS = Contact Sensor, MD = Motion Detector.

Pour des raisons pratiques, notre approche n'a pas pu être déployée au commencement de l'expérimentation DomAssist. Notre modèle a donc été appliqué rétrospectivement pour vérifier la conformité du domicile de chaque participant en exécutant les règles sur les logs accumulés durant l'expérimentation.

En étudiant les conditions de l'expérimentation et nos rôles, nous avons spécifié les règles de conformité qui rendent explicites les conditions de l'étude : les participants vivent seuls (règle de non omniprésence) et certaines interactions avec l'environnement doivent suivre un motif spécifique (porte restée ouverte). D'autres règles ne dépendent pas de l'objectif de l'étude et peuvent être généralisées. La règle d'inclusion de présence et ses raffinements (intersection de présence et besoin de présence) en sont des illustrations. Examinons maintenant ces règles.

Non omniprésence. Cette règle assure qu'un rôle de présence

n'est pas détecté simultanément à deux localisations différentes. En pratique, en fonction de la réactivité des capteurs de mouvements utilisés pour la détection de présence, quelques brefs chevauchements peuvent survenir et doivent être ignorés. Typiquement, un détecteur de présence signale une absence avec une latence. Cette latence rend possible la détection simultanée de l'utilisateur dans deux pièces pendant quelques instants.

Porte restée ouverte. Cette règle assure que la période d'un rôle d'ouverture de porte ne dure pas plus d'un certain temps (trois heures dans notre configuration). Notons que cette règle est exécutée sur les logs d'évènements, en complément d'autres applications d'assistance qui peuvent réagir aussi à une telle situation. Par exemple, DomAssist inclut une application qui surveille la porte d'entrée. Cette application notifie l'utilisateur quand la porte est restée ouverte, sans surveillance pendant quelques minutes (la durée est configurée en fonction de l'utilisateur). Aussi, quand elle est appliquée aux logs de notre étude, cette règle détecte dans la plupart des cas des problèmes d'installation. Cette situation concerne également les portes non surveillées par des applications de sécurité (telle une porte de placard), ce qui s'explique par le fait que les participants sont routinisés dans leurs activités [15] et n'ont pas déclinés cognitivement de façon significative durant l'étude.

15. Valérie BERGUA et al. [2013]. "Restriction in instrumental activities of daily living in older persons : Association with preferences for routines and psychological vulnerability". In : *The International Journal of Aging and Human Development* 77.4, p. 309–329.

Inclusion de présence. Chaque pièce pour laquelle une interaction doit être détectée est équipée avec un détecteur de mouvement. Par conséquent, nous avons généralisé la règle de présence dans la cuisine à toutes les pièces : toute interaction, qui n'est pas une présence, à une localisation donnée, doit être inclue dans un rôle de présence à la même localisation.

Intersection de présence. L'inclusion de présence peut être trop contraignante pour s'appliquer à certaines situations. Parfois, nous devons simplement nous assurer que la présence et d'autres interactions ont une intersections non nulle quand elle sont situées dans la même pièce. Une telle règle s'applique dans l'entrée du domicile, équipée d'un capteur de contact sur la porte d'entrée et un capteur de mouvement dans l'entrée (zone située à l'intérieur du domicile). En effet, si l'utilisateur ouvre la porte, que ce soit depuis l'extérieur ou depuis l'intérieur, les rôles de présence et d'ouverture ont une période de temps durant laquelle leur intersection est non-nulle.

Besoin de présence. Pour s'assurer que le détecteur de mouvement est toujours actif même si il est mal positionné, nous introduisons la règle suivante : chaque rôle, qui n'est pas une présence, doit être accompagné d'un évènement de rôle présence

à la même localisation. Cette présence doit arriver au même moment plus ou moins dix minutes. Cette règle rend explicite le fait que le capteur de mouvement est toujours couplé avec un ou plusieurs autres capteurs dans notre configuration. La violation de cette règle indique principalement que le capteur de mouvement ne fonctionne pas correctement, probablement car il est toujours enregistré dans le système, mais n'émet plus aucune donnée.

Méthodologie

Nous avons collecté les logs des capteurs placés dans les habitations des 24 participants à l'étude. Ces participants sont âgés de 80 ans en moyenne et habitent seuls. Les données collectées couvrent une période de neuf mois. Cependant, des problèmes techniques (*e.g.*, accès internet, passerelle domotique, serveur) nous ont poussé à ignorer certaines périodes de temps ; ces problèmes peuvent être directement détectés par la plate-forme. Les logs de l'expérimentation ont également été nettoyés en éliminant les évènements de rôle non conformes pouvant être détectés par un mécanisme de tolérance aux fautes détectant les fautes de capteurs et de réseau, selon la classification définie par Chetan *et al.* [Chetan et al. 2005]. Par exemple, l'absence de signal de heartbeat, normalement émis par tout capteur, est détectée par les couches basses de la plate-forme et signalée comme un échec de communication avec le capteur.

Nous avons examiné le plan du domicile de chaque participant, spécifiant le placement et le positionnement des capteurs. Ce document a été utilisé pour diagnostiquer les problèmes quand des violations de conformités sont arrivées dans les logs. D'autres ressources ont été également disponibles pour diagnostiquer les violations, telles que les fichiers de suivi des interventions remplies par les professionnels chargés de faire passer des questionnaires à chaque participant durant l'étude.

Résultats expérimentaux

Le modèle défini pour DomAssist permet de lever deux types d'anomalies : (1) *les non-conformités permanentes* – elles font réfèrence à une règle systématiquement transgressée, indiquant des non concordances permanentes entre l'infrastructure et le modèle ; (2) *les non-conformités émergentes* – elles correspondent à une règle qui est vérifiée la plupart du temps, mais échoue ponctuellement.

Non-conformités permanentes. Dans un domicile, l'interaction d'ouverture est détectée pour la porte du placard de la cuisine,

mais les règles *inclusion de présence* et *intersection de présence* échouent systématiquement. Cependant, la règle de *besoin de présence* n'échoue jamais, indiquant que la porte de placard est ouverte mais jamais couverte, ni même en intersection avec une interaction de présence dans la cuisine ; la présence est donc détectée à des moments proches mais disjoints de l'interaction d'ouverture. La consultation des documents relatifs à cette installation ont montré que ce placard n'est pas localisé dans la cuisine mais dans une pièce attenante, comme montré en Figure 4. Cette situation montre un problème dans la définition du modèle pour cette installation. Par conséquent, nous avons modifié le modèle de cette installation en y supprimant les règles d'inclusion de présence et d'intersection de présence. Nous les avons remplacées par la règle besoin de présence, mieux adaptée à cette configuration.

Dans ce même domicile, un autre problème à été identifié : la règle *inclusion de présence* échoue systématiquement sur le rôle d'ouverture du frigidaire, mais l'*intersection de présence* n'échoue jamais. Bien que le frigidaire soit localisé dans la cuisine, le capteur de mouvement utilisé pour mesuré la présence dans la cuisine, n'a pas été positionné pour couvrir l'ensemble de la cuisine, comme montré en Figure 4. Ce problème provient dans ce cas d'une erreur à l'installation et la solution consiste simplement à changer le positionnement du capteur de mouvement.

Nous avons observé la même configuration du placard de la cuisine situé en dehors de la cuisine, dans deux autres domiciles.

Généralement, ces types de problèmes surviennent quand la plate-forme d'assistance est déployée à large échelle. Dans ce contexte, les installations sont faites par un professionnel, et non par les chercheurs qui ont conçu la plate-forme et qui possèdent un savoir implicite sur la bonne manière de déployer. Dans notre cas, même pour 24 domiciles, quelques installations ont été faites par des membres n'ayant pas ce savoir, et donc n'ont pas pu se conformés à certaines règles implicites.

Si notre outil avait été utilisé dès la phase de déploiement, il aurait été possible de détecter les anomalies d'installation dès le départ, pendant l'installation ou dans les premiers jours d'exploitation. Par ailleurs, une fois déployées, des règles supplémentaires peuvent affiner les modèle pour prendre en compte des spécifications imprévues (*e.g.*, cuisine en forme de L). Une fois l'installation ou le modèle affiné, notre approche contribue à la détection d'anomalies au quotidien, en une période de fonctionnement normal.

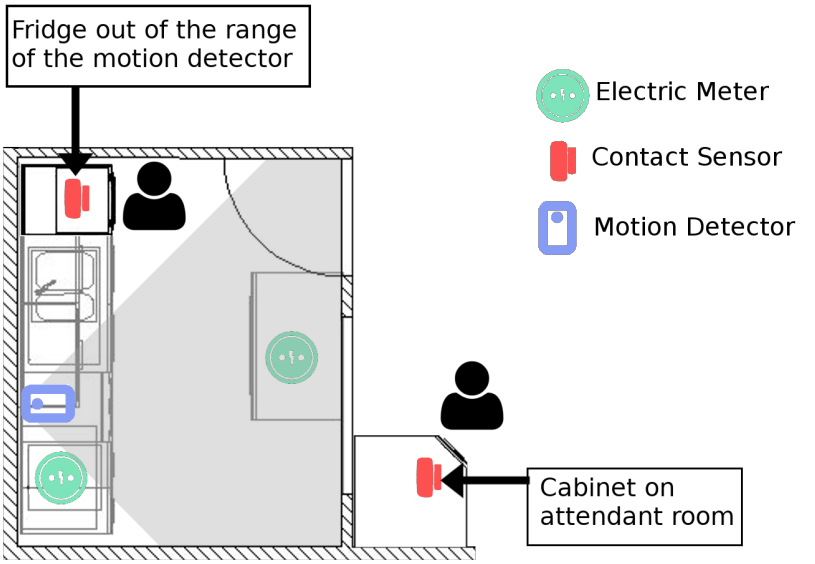


FIGURE 4: Illustration de problèmes d'installation.

Non-conformités émergentes. Un motif de non concordance a été observé dans cinq domiciles à différentes périodes de temps. Pendant plusieurs jours, la règle *inclusion de présence* a été enfreinte par une absence de présence dans la cuisine causée par un capteur de mouvement qui dysfonctionnait. Dans chacun des cas, les règles d'*intersection de présence* et de *besoin de présence* étaient également transgressée durant ces périodes. Cette dernière règle montre qu'aucun rôle de présence n'a été reconnu durant les 20 minutes entourant cette violation. Les résultats de ces règles fournissent de précieuses informations pour trouver la cause des dysfonctionnement relevés. Il est alors raisonnable de supposer que la cause de cet échec des règles est un dysfonctionnement temporaire du rôle de présence. Nous pouvons supposer que le capteur de mouvement a pu être obstrué ou déplacé.

Une situation similaire est arrivée dans l'entrée de deux autres domiciles : la porte a parfois été ouverte, sans qu'aucune présence n'ait été détectée.

La *porte restée ouverte* a été observée dans quatre domiciles à différents endroits comme le frigidaire, le placard de la cuisine ou la penderie (*i.e.,* la porte est restée ouverte plus de trois heures). D'après les fichiers de suivi des interventions de l'expérimentation, les utilisateurs concernés ont été questionnés après quelques jours sur les comportements erronés des applications reposant sur ces interactions. Ils ont indiqué que les capteurs de contact étaient tombés. Les installations ont alors été réparées par les techniciens chargés des interventions. Notre modèle, si il avait été disponible durant cette ex-

périmentation aurait permis de rapporter ces incidents, et de les solutionner plus rapidement. Cette réactivité est indispensable pour la pertinence des résultats des application sensibles au contexte. Elle fait toute la différence entre une application utile et une application qui harcèle l'utilisateur avec des notifications non pertinentes.

Discussion

Un modèle explicite d'une infrastructure est capable de détecter les dysfonctionnements du systèmes durant la phase d'installation ou pendant l'exploitation normale. Comme suggéré par certains de nos exemples, une fois une erreur détectée, quelques techniques de diagnostic peuvent être utilisées pour identifier les rôles qui sont la source de l'erreur.

Premièrement, si plusieurs règles sont violées en même temps, et que ces règles concernent des ensembles de capteurs qui se chevauchent, il est alors probable que le rôle à la source de cette erreur soit à rechercher à l'intersection de ces ensembles. Par exemple, quand la règle *intersection de présence* échoue en même temps pour le rôle présence dans la cuisine et pour différentes interactions qui ne sont pas des présences (frigidaire, placard, *etc.*), le rôle défaillant est probablement la présence dans la cuisine.

Deuxièmement, concevoir une version raffinée d'une règle qui vérifie un ensemble de capteurs (ou est un sous-ensemble de la règle de base) peut s'avérer utile pour diriger la recherche d'un rôle défaillant ou d'un capteur qui dysfonctionne. Cette idée est illustrée dans notre modèle pour DomAssist. Il y a une chaîne de trois règles R_i avec des conditions de moins en moins précises : *inclusion de présence*, *intersection de présence* et *besoin de présence*. Toutes ces règles sont du type $p \rightarrow q_i$ pour $(i = 1..3)$ où le postulat p est identique, mais la conclusion q_i est de plus en plus faible. Il y a une implication tout au long de la chaîne, $R_1 \rightarrow R_2 \rightarrow R_3$; ou inversement, quand une des règles échoue, les règles les plus fortes échouent également. Basé sur l'analyse des règles, nous pouvons définir un arbre de décision comme celui en Figure 5 pour aider au diagnostic.

L'utilité de notre approche pour assurer continuellement la conformité d'une infrastructure va au-delà de la détection d'infrastructures défaillantes. Par exemple, un modèle d'infrastructure pourrait fournir un modèle de référence pour des applications disponibles dans un catalogue en ligne de type Appstore : une application pourrait être installée uniquement lorsqu'elle est conforme au modèle d'infrastructure. En pra-

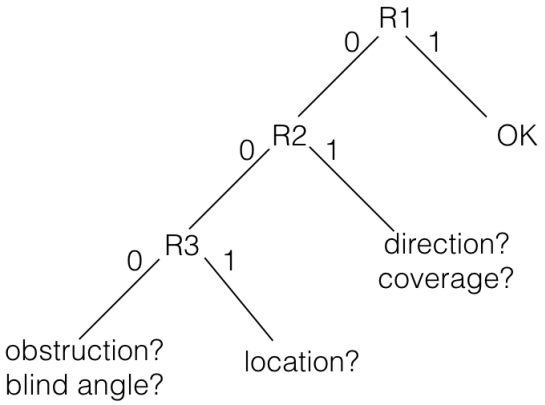


Figure 5: Arbre de décision binaire aidant au diagnostic de la source d'une défaillance.

tique, ce filtrage des applications au moment de l'installation pourrait remplacer un certain nombre de tests qui sont nécessaires aujourd'hui pour chaque application déployée sur une installation donnée, pourvu que le modèle couvre toutes les hypothèses requises par les applications.

Conclusion

Nous avons montré que les applications sensibles au contexte reposent fréquemment sur des suppositions implicites concernant l'infrastructure de capteurs. Ces suppositions se traduisent généralement en terme de programmation par des instructions conditionnelles qui polluent le code avec des préoccupations non fonctionnelles. Cette approche de programmation défensive peut être évitée en exprimant ces suppositions, non pas dans le code des applications, mais en les factorisant dans un modèle explicite d'infrastructure de capteurs. Non seulement la violation d'une règle du modèle d'infrastructure permet d'alerter rapidement sur un dysfonctionnement de l'installation, mais cette situation contribue également à diagnostiquer le problème. Une telle approche permet de s'assurer de la fiabilité de la sensibilité au contexte des applications d'assistance domiciliaire. De plus, la définition du modèle permet de fourmuler des instructions détaillées quant à l'installation des capteurs dans le domicile, rendant ce dernier sensible au contexte.

4
Analyse de domaine

Le chapitre précédent nous donne une fondation solide pour déployer des services contextuels dans une maison connectée, en assurant la fiabilité de son infrastructure de capteurs. Nous pouvons désormais nous intéresser au développement de ces services contextuels. Pour cela, nous allons analyser dans ce chapitre un large éventail de services d'assistance domiciliaire dédiés aux personnes âgées, en considérant la variété des besoins des intervenants. Cette analyse permet d'identifier les concepts communs et les variations entre différentes couches de services déployés et utilisés au quotidien, pour en extraire des concepts clés et des opérations spécifiques aux traitements sensibles au contexte [1].

1. Ce travail à fait l'objet d'une soumission : Adrien Carteron et al. [2018]. "A Domain-Specific Approach to Unifying the Many Dimensions of Context-Aware Home Service Development". In : *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom) (PerCom 2018)*. Athens, Greece.

Sommaire

Aperçu

— *DomAssist* Présentation d'une expérimentation écologique large échelle sur laquelle nous nous appuyons pour notre analyse.

— *Analyse de domaine* Identification des concepts clés, des opérations spécifiques et des besoins communs aux services sensibles au contexte.

Les services d'assistance dédiés au maintien à domicile des personnes âgées sont encore un domaine émergeant et le chemin vers l'adoption reste un sujet d'étude [2]. La littérature comporte encore peu d'articles concernant le déploiement de solutions d'assistance dans de vrais domiciles [3]. Cependant, nous avons pu faire levier sur le projet DomAssist pour conduire notre analyse du domaine.

2. Jeffrey Kaye [2017]. "Making Pervasive Computing Technology Pervasive for Health & Wellness in Aging". In : *Public Policy & Aging Report*

3. Jeffrey A Kaye et al. [2011]. "Intelligent systems for assessing aging changes : home-based, unobtrusive, and continuous assessment of aging". In : *Journals of Gerontology Series B : Psychological Sciences and Social Sciences* 66.suppl_1, p. i180–i190

Enjeux d'une expérimentation écologique à large échelle

L'étude pilote de la plate-forme, DomAssist, présentée dans le Chapitre 3, a donné des résultats encourageants quant à l'acceptabilité de la technologie, par un groupe de 24 utilisateurs, âgés de 80 ans en moyenne, pendant une durée de 9 mois. D'un point de vue technologique, cette expérimentation, était une première confrontation aux difficultés et contraintes d'un déploiement écologique. Nous avons donc conçu l'approche présentée dans le Chapitre 3 pour nous assurer du bon fonctionnement de l'infrastructure de capteurs, lors de son installation et durant son exploitation, et pour garantir la consistance des services d'assistance.

Faisant suite à l'étude pilote, une nouvelle expérimentation d'autonomie domiciliaire des personnes âgées est en cours à plus large échelle, couvrant un plus grand nombre d'utilisateurs pour une durée de 12 mois [4]. La plate-forme est actuellement déployée chez 129 personnes vivant seules et âgées de 82 ans en moyenne [5]. Cette expérimentation est construite en lien étroit avec les acteurs du domaine de l'autonomie domiciliaire des personnes âgées. Ainsi, la conception et la mise en place des services ont impliqué tous les acteurs : utilisateurs, aidants, ergothérapeutes, psychologues, experts en facteurs humains, techniciens d'installation et de maintenance et informaticiens. Le modèle d'assistance qui en résulte offre des services dédiés à des tâches variées comme les rappels de rendez-vous, la surveillance d'activités du quotidien ainsi que leur rappel si elles ne sont pas effectuées, la sécurisation de l'utilisateur, ou encore le bilan quotidien des activités. L'analyse des données recueillies permet d'évaluer et d'adapter l'assistance en fonction de chaque utilisateur et de ses dégradations éventuelles.

4. http://phoenix.inria.fr/research-projects/homeassist-500

5. Charles Consel et al. [2017]. "HomeAssist : An Assisted Living Platform for Aging in Place Based on an Interdisciplinary Approach". In : *Proceedings of the 8th International Conference on Applied Human Factors and Ergonomics (AHFE 2017)*. Springer

Cette étude à large échelle offre de nombreux avantages pour construire notre analyse du domaine : (1) elle est déployée dans des environnements réels ; (2) elle constitue un soutien pour le maintien à domicile de personnes âgées avec des utilisateurs fragiles ayant des besoins immédiats. La précédente expérimentation avait pour but principal d'ajuster et de

valider la plate-forme DomAssist en tant que plate-forme d'assistance domiciliaire, en n'incluant pas d'utilisateurs avec un statut fonctionnel trop dégradé. La nouvelle étude est ouverte à plus d'utilisateurs et permet donc d'élargir les services d'assistance. (3) L'étude conduite est suffisamment longue pour que les problèmes de maintenance et d'évolution doivent être traités avec réactivité. (4) La plate-forme est déployée à une échelle suffisamment large pour que l'administration des domiciles ait besoin d'être supportée par des services. Avec un tel nombre de domiciles installés, il est indispensable de disposer de services qui signalent automatiquement les cas de dysfonctionnements, afin que des dispositions soient prises au plus tôt. (5) En conséquence, les services existants reflètent un large éventail de besoins exprimés par les intervenants. Nous disposons maintenant non seulement d'une large gamme de services d'assistance domiciliaire, mais également de services relatifs à la maintenance et la supervision de l'infrastructure de cette assistance.

Des services d'assistance

Une étude de besoins, conduite auprès de personnes âgées vivant seules et de leurs aidants, a permis aux experts en vieillissement, psychologues et ergothérapeutes de formuler un cahier des charges des services d'assistance à développer et au degré de personnalisation indispensable au fonctionnement de ces services. Comme évoqué plus haut, la plate-forme propose trois catégories de services d'assistance : le support des activités du quotidien, la sécurité de l'utilisateur et ses interactions sociales [6]. Les services d'assistance sont développés en Java en utilisant une méthodologie de conception outillée basée sur la paradigme *Sense Compute Control* [7] [8].

Les services sont disponibles dans un catalogue d'applications en ligne à la manière de ce qui est proposé de nos jours pour les smartphones. Il est alors aisé d'adapter l'assistance fournie par la plate-forme en supprimant ou ajoutant des applications en fonction de l'évolution des besoins de l'utilisateur. Les applications peuvent également être personnalisées en fonction de l'utilisateur. Par exemple, un paramètre d'installation permet de définir la durée de l'ouverture de la porte d'entrée sans surveillance, avant qu'une alerte soit envoyée.

Infrastructure

La plate-forme DomAssist se compose d'une architecture client-serveur. Le serveur instancie autant de machines virtuelles

6. Consel et al. 2017

7. Benjamin Bertran et al. [2014]. "DiaSuite : A tool suite to develop Sense/Compute/Control applications". In : *Science of Computer Programming* 79, p. 39–51.

8. Damien Cassou et al. [2012]. "Toward a tool-based development methodology for pervasive computing applications". In : *IEEE Transactions on Software Engineering* 38.6, p. 1445–1463.

qu'il y a de domiciles équipés de DomAssist. Chaque machine virtuelle permet d'exécuter les services d'assistance sélectionnés par l'utilisateur et les aidants. Les services d'assistance sont alimentés avec les données de mesure d'interactions via Internet et une passerelle domotique déployée dans le domicile installé. La passerelle centralise les informations en provenance des capteurs, installés aux endroits stratégiques du domicile pour surveiller les activités. La passerelle relaie également les actions commandées par les services à destination des actionneurs. Cette architecture est illustrée en Figure 6.

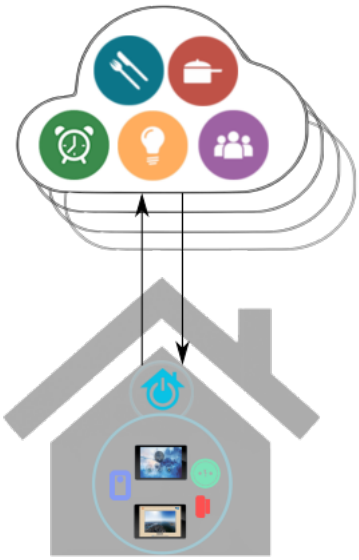


FIGURE 6: Illustration de l'architecture de la plate-forme DomAssist.

Dans l'étude DomAssist, un domicile typique est pourvu de quatre capteurs de contact (porte d'entrée, frigidaire, armoire, *etc.*), six détecteurs de mouvements (zone de l'entrée, cuisine, salle de bain, *etc.*), et deux capteurs de consommation électrique, qui peuvent également allumer et éteindre les appareils qui y sont branchés (chemin lumineux, micro-onde, machine à café, *etc.*). Le nombre et le type de capteurs/actionneurs peut varier en fonction de la configuration du domicile et des activités à surveiller. Enfin, le domicile est équipé avec deux tablettes. La première, une tablette fixe, est placée à un endroit central dans le domicile et est toujours alimentée électriquement. Cette tablette est dédiée aux interactions de la plate-forme avec l'utilisateur via les notifications émises par les applications d'assistance, qui alertent l'utilisateur d'une situation donnée (*e.g.*, porte d'entrée ouverte et non surveillée depuis un certain temps)[9]. La seconde tablette, dont la mobilité est autorisée, est utilisée pour les activités sociales. Elle dispose notamment d'une application de gestion simplifiée du courrier électronique, avec synthèse vocale. D'autres applications de communication et des jeux collaboratifs sont également installées. La Figure 7 montre un exemple de domicile équipé.

9. Charles CONSEL et al. [2015]. "A Unifying Notification System To Scale Up Assistive Services". In : *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*. ACM, p. 77–87

Scénarios pour le maintien à domicile

Les services proposés par les applications disponibles sur la plate-forme DomAssist concernent l'assistance domiciliaire. Cependant, il existe de nombreux autres services nécessaires au fonctionnement de la plate-forme, notamment sa maintenance (*e.g.*, infrastructure de capteurs, infrastructure logicielle, réseau, *etc.*), qui vont influencer le bon fonctionnement des services d'assistance. Ces services *annexes* sont instanciés, non pas directement au sein de la plate-forme DomAssist en tant qu'applications disponibles, mais avec différents framework spécialisés (*e.g.*, surveillance de la communication réseaux, charge des

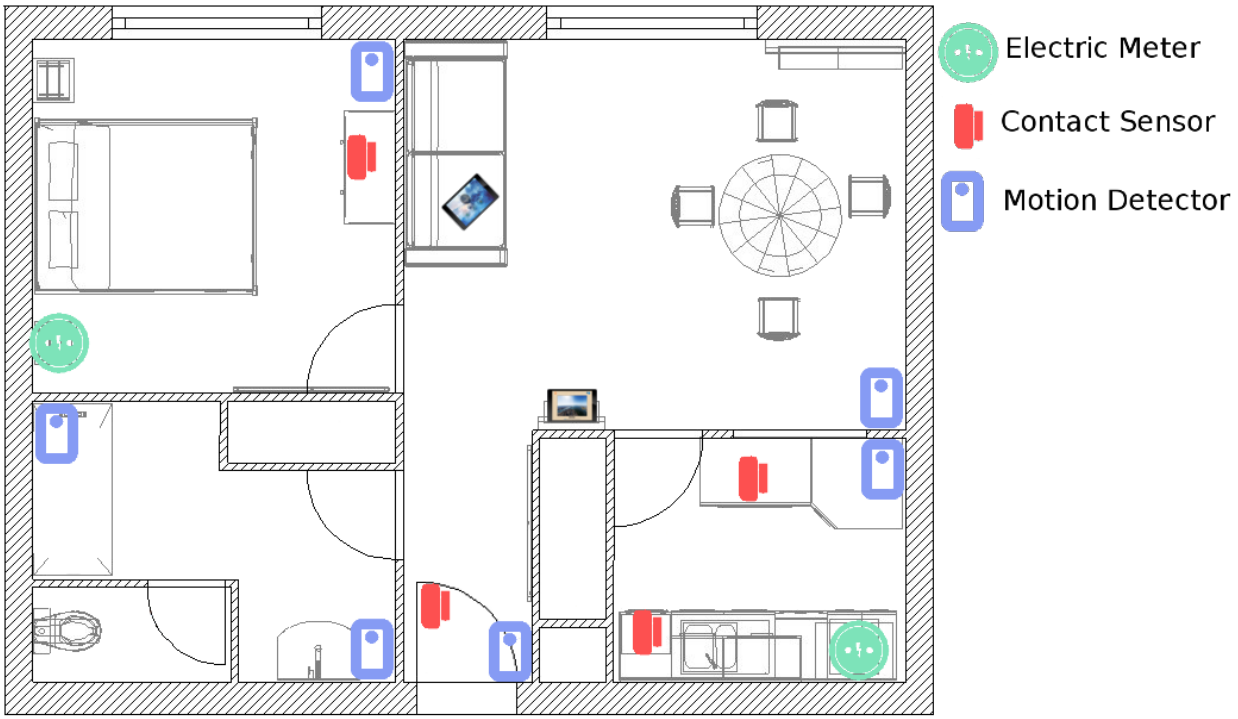


FIGURE 7: Exemple de domicile déployé avec son installation de capteurs.

serveurs, état des batteries des capteurs, état de l'infrastructure de capteurs, *etc.*).

Pour définir quels sont les besoins des intervenants en matière de services, nous avons recours à la plate-forme d'assistance déployée à large échelle, aux intervenants ayant contribué à la définition de services d'assistance, et aux intervenants participant à son exploitation. À partir de cet ensemble d'expertise, nous décrivons dans la Table 2, quelques scénarios illustrant la variété d'intervenants et la variété de préoccupations nécessaires au maintien à domicile de personnes âgées, lorsque des services d'assistance sont sensibles au contexte.

Stakeholder	**Domain**	**Name**	**Description**
Older adult	Safety	Door Alert	Entrance door is open and is unattended for 5 minutes
Caregiver	Daily Activities	Reheating A Frozen Meal	Freezer gets used and stove gets turned on within 10 minutes or Freezer gets used during stove is on, during lunch time (or dinner time)
Home Technician	Maintenance	Presence Dependency	The cupboard gets opened in the kitchen, while a presence in the kitchen is false
Home Technician	Maintenance	Communication Failure	A sensor fails to communicate for 24 hours

TABLE 2: Example de scénarios de services d'assistance.

Le premier scénario concerne la sécurité de l'utilisateur. Ce

scénario surveille la porte d'entrée pour s'assurer qu'elle ne reste pas ouverte trop longtemps sans être surveillée. Le deuxième scénario implique à la fois l'utilisateur et l'aidant. Il illustre un besoin exprimé par les aidants, en fournissant un service qui rappelle à l'utilisateur une routine du quotidien, dans le cas où il ne l'aurait pas réalisée (dans l'exemple, la préparation du repas). La formulation du scénario est déterminée par l'utilisateur qui va définir lui-même la façon dont il effectue ses routines quotidiennes. Les deux derniers scénarios concernent des besoins exprimés par les techniciens pour garder opérationnels les domiciles sensible au contexte. Le premier scénario de maintenance détecte l'utilisation d'un placard sans que celle-ci soit recouverte par la détection d'une présence. Le second scénario de maintenance détecte quand la passerelle échoue à communiquer avec un capteur ; cette situation se produit lorsque le capteur n'a plus de batterie ou dysfonctionne.

Ces scénarios offrent un aperçu des types de services sensibles au contexte nécessaires pour le maintien à domicile de personnes âgées. Ces services fournissent directement une assistance ou servent de support au bon fonctionnement de la plate-forme. Certains services tels que "Door Alert", peuvent convenir à la plupart des utilisateurs. En revanche, d'autres services, typiquement les services concernant les activités du quotidien, nécessitent un certain degré de personnalisation pour être efficaces. Ceci est illustré par l'activité de préparation de repas et le scénario "Reheating a Frozen Meal".

De même, "Communication Failure" peut s'appliquer à n'importe quel domicile sensible au contexte, alors que "Presence Dependency" demande d'instancier les règles en fonction de la localisation des capteurs.

Il est à noter que la formulation de certains scénarios, principalement les scénarios de sécurité de la personne ou les scénarios de maintenance, décrit des situations anormales. Par exemple, pour le scénario de "Presence Dependency", la situation normale serait décrite ainsi : *"Quand le placard est est ouvert dans la cuisine, la présence dans la cuisine est vraie"*. Or, dans ces cas là, le but de ces scénarios est d'identifier les situations inhabituelles, c'est donc ces situations qu'ils doivent décrire.

Une autre observation frappante est que certains services de maintenance reprennent directement des règles qui faisaient partie du modèle d'infrastructure des capteurs, décrit dans le Chapitre 3, sauf qu'elles sont maintenant formulées par la négative, pour chercher les situations non-désirées, comme expliqué ci-haut. En fait, *toutes* les régles du modèle explicite d'infrastructure peuvent être ainsi exprimées comme des services d'acquisition de contexte, où le contexte concerne alors les dys-

fonctionnements de l'infrastructure. Ainsi, le spectre des services contextuels dans une maison connectée inclut comme un sous-ensemble le modèle d'infrastructure, dans le spectre bien plus large des services destinés à tous les intervenants de l'assistance domiciliaire.

Analyse de commonalités et variabilités

Les scénarios définis nous permettent de faire une étude de commonalités et de variabilités.

Commonalités

Tous les services se réfèrent à une notion d'*environnement* dans lequel sont effectuées les mesures. Ces mesures observent des interactions qui peuvent se passer soit dans l'environnement physique (*e.g.,* un mouvement détecté), soit dans l'environnement numérique (*e.g.,* un rappel d'évènement délivré par un agenda). Concernant ces mesures de l'environnement, nous avons identifiés deux concepts récurrents dans les scénarios étudiés : les évènements et les états. Un *évènement* définit une mesure de l'environnement au moment où l'interaction correspondante se produit (*e.g.,* une porte *devient* ouverte/fermée) – les évènements sont soulignés avec des pointillés dans les scénarios de la Table2. Un *état* exprime l'accès au statut courant d'une mesure préalable de l'environnement (*e.g.,* la porte *est* ouverte/fermée) – les états soulignés avec une ligne pleine dans la Table 2. Ces deux concepts d'état et d'évènement permettent donc d'exprimer soit le moment où une interaction se produit, pour les évènements, soit le statut d'une interaction déjà produite qui persiste dans le temps, pour les états.

Il est possible également d'identifier dans les scénarios, des concepts de *composition*. Spécifiquement, la combinaison de mesures d'environnement peut définir un *ordre* dans lequel les interactions doivent arriver et la *durée*, tant des interactions que des compositions – ces contraintes sont soulignées avec des points dans la Table 2.

Variabilités

Les mesures environnementales peuvent être réalisées par une variété d'entités, matérielles (*e.g.,* capteurs), logicielles (*e.g.,* agenda), locales (*e.g.,* porte), distante (*e.g.,* nouvel email), *etc.* Le niveau d'abstraction varie énormément selon les mesures environnementales. Par exemple, un évènement peut être produit par un capteur, sitôt qu'un mouvement est détecté dans

l'entrée. Parallèlement, cette même interaction peut servir pour être agrégée avec d'autres évènements et produire un évènement "rentrée au domicile". Plusieurs contraintes d'ordre sont possibles pour la composition des interactions. Une interaction peut en *précéder* une autre, une interaction peut se produire *durant* une autre, et une interaction peut en *chevaucher* une autre. Cependant, toutes ces contraintes ne sont pas applicables à tous les types d'interactions (*i.e.,* évènements et états). Par exemple, seulement deux états peuvent se chevaucher, alors que les évènements ne peuvent pas se chevaucher à cause de leur nature ponctuelle.

Concepts spécifiques au domaine

Suivant cette analyse, nous pouvons d'ores et déjà exprimer certains concepts. Nous retrouvons un modèle d'environnement, comme évoqué en Section 3.1, avec des informations sur l'interaction (le type d'interaction mesurée, sa localisation, la valeur mesurée, la positionnement dans le temps) et les niveaux d'abstraction d'une interaction (*e.g.,* physiques, logiciels, *etc.*). Il est maintenant possible de définir plus précisément le concept d'évènement et d'état :

Évènement : un changement de valeur, pour une entité mesurée, qui se produit à un moment donné.

État : Une valeur qui subsiste pendant une période. L'état débute à un moment donné et finit à un autre moment.

Pour composer ces états et évènement dans le temps, nous avons extrait de notre analyse des opérateurs de composition (*e.g.,* précédence, chevauchement, *etc.*).

Besoins communs

L'analyse des services d'une plate-forme d'assistance domiciliaire met en évidence certaines contraintes indispensables à l'exploitation des services. Du fait de leur sensibilité au contexte, et leur objectifs parfois critiques dans l'assistance domiciliaire (*e.g.,* sécurité, maintenance, *etc.*), les services doivent être capables de réagir rapidement aux mesures d'environnement qu'ils traitent.

De plus, des services avec des objectifs différents doivent pouvoir accéder aux mêmes données pour effectuer des traitements différents. La variété d'intervenants et la variété de leur expertise impliquent une forte hétérogénéité de données qu'il faut exprimer en terme d'état et d'évènements.

Les services expriment un ensemble d'interactions qui peut

revenir de façon récurrente dans le flux d'évènements d'interactions. Par exemple, les évènements décrivant l'activité de préparation d'un repas reviennent typiquement tous les jours. Par conséquent, un service vérifiant que l'activité à été effectué doit être testé de manière récurrente.

Comme observé, il y a une forte composante temporelle dans l'expression des services. Tant pour expliciter la durée de la persistance d'un état, que pour exprimer de simples relations entre états/évènements et la durée de cette relation, cette composante temporelle doit être partie intégrante de l'expressivité des services sensibles au contexte.

La variété d'intervenants implique également autant de manières de formuler un scénario de service. Les scénarios exprimés par les intervenants doivent couvrir l'ensemble des services nécessaires à une plate-forme d'assistance de maintien à domicile des personnes âgées. Il faut donc être capable de proposer une solution permettant d'exprimer ces services en couvrant leur spectre d'expertise et d'objectifs. Cependant, malgré l'hétérogénéité des données captées et malgré la large variété des besoins entre les intervenants des différents domaines d'expertise, ces services sensibles au contextes agencent des concepts communs qui permettent d'uniformiser leur formulation et leur traitements.

5
Maloya : Un langage spécifique dédié aux services sensibles au contextes

Ce chapitre présente notre approche dédiée au domaine de l'assistance domiciliaire. Dans un premier temps, nous décrivons l'architecture logicielle sous-jacente à notre approche. Ensuite, nous introduirons un langage dédié pour développer des services sensibles au contexte .

Sommaire

Contributions

— *Architecture logicielle* Un architecture qui couvre les besoins d'unification de données hétérogènes et d'exécuter des services en continu sur le flux d'évènements.

— *Langage dédié* Un langage spécifique pour développer un large spectre de services sensibles au contexte, avec un haut niveau d'asbtraction et un traitement uniforme de sources de données hétérogènes.

— *Validation* redéfinition de 55 services, couvrant les besoins de tous les intervenants d'une plate-forme d'assistance pour le maintien à domicile de personnes âgées.

Sur la base de l'analyse de domaine présentée dans le chapitre précédent, nous somme en mesure de proposer un langage dédié aux services sensibles au contexte, appelé Maloya, ainsi que l'architecture logicielle sur laquelle repose ce langage :

— Notre architecture logicielle est *centrée données*, en unifiant des sources hétérogènes de données, que ce soit en provenance de capteurs matériels ou de capteurs logiciels. Elle fournit à l'ensemble des services une vue canonique des données mesurées. Ainsi, cette forme canonique couvre d'une part, les services de maintenance requérant l'état bas niveau des capteurs, et d'autre part, les services spécifiques aux aidants nécessitant des mesures d'activités de haut niveau.

— Le langage dédié Maloya fournit à la fois un cadre conceptuel et des outils pour concevoir et développer des services domiciliaires pour les personnes âgées. Notre langage est *orientée données* avec des services définis en terme de règles traitant des évènements et des états, et des opérateurs pour les combiner.

Une approche dédiée au domaine de l'assistance domiciliaire

Nous présentons ici les étapes principales de notre approche spécifique au domaine pour développer des services sensibles au contexte dédié au maintien à domicile des personnes âgées.

La fourniture d'un service sensible au contexte, selon notre approche, implique deux phases principales, illustrées en Figure 8 :

— La phase de développement : les intervenants définissent un service sous forme de règle en langage Maloya. Cette règle métier est ensuite compilée vers une règle de plus bas niveau en langage de traitement d'évènements. Les outils assistant le développeur de service lors de cette phase comprennent le compilateur et une interface de gestion des services.

— La phase d'exploitation : les services compilés sont exécutés sur les flux d'évènements provenant d'une maison sensible au contexte. Une fois mise en place, une règle est exécutée de façon à révéler chaque occurrence de la séquence d'évènements qu'elle décrit, jusqu'à ce que la règle soit supprimée, ou mise à jour. La phase d'exploitation est implantée par un analyseur d'évènements d'interaction qui transforme ces évènements en une forme canonique appelée *StreamEvent*

(ou simplement, flux d'évènement, par la suite) pour alimenter un moteur de traitement d'évènements complexes (CEP). Ce moteur CEP est chargé de l'exécution des règles compilées sur le flux d'évènements.

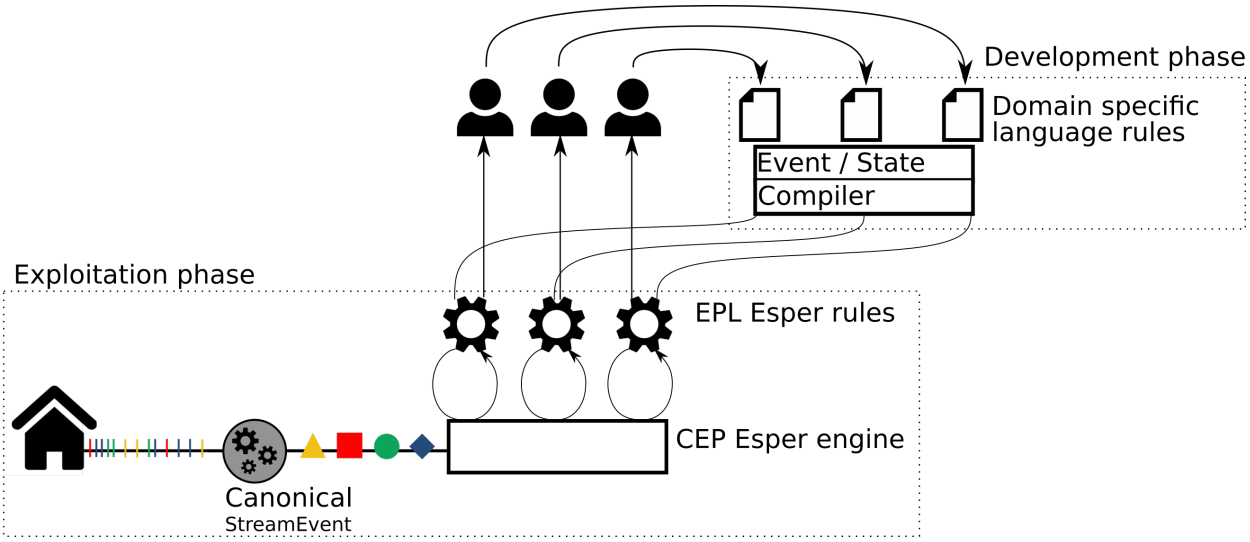


FIGURE 8: Vue globale de notre approche dédiée au domaine.

Définition du service

La première étape est initiée par les intervenants qui expriment les scénarios de services. Ces scénarios sont soit directement écrits dans notre langage dédié par l'intervenant en exploitant les concepts d'états/évènements et les opérateurs de composition disponibles, si ils ont le bagage nécessaire, soit par un développeur de services, dans le cas contraire.

Notre implantation offre une interface graphique simple pour définir, compiler, supprimer, et mettre à jour un service, comme illustré en Figure 9.

Compilation du service

Les services de haut niveau sont compilés vers un langage de traitement d'évènements qui exprime des règles de plus bas niveau. Le langage cible de traitement d'évènements ne supporte pas directement la notion d'état ainsi que nos opérateurs. Lors des étapes de compilation, ces concepts sont donc explicités sous forme de combinaisons d'évènements correspondants.

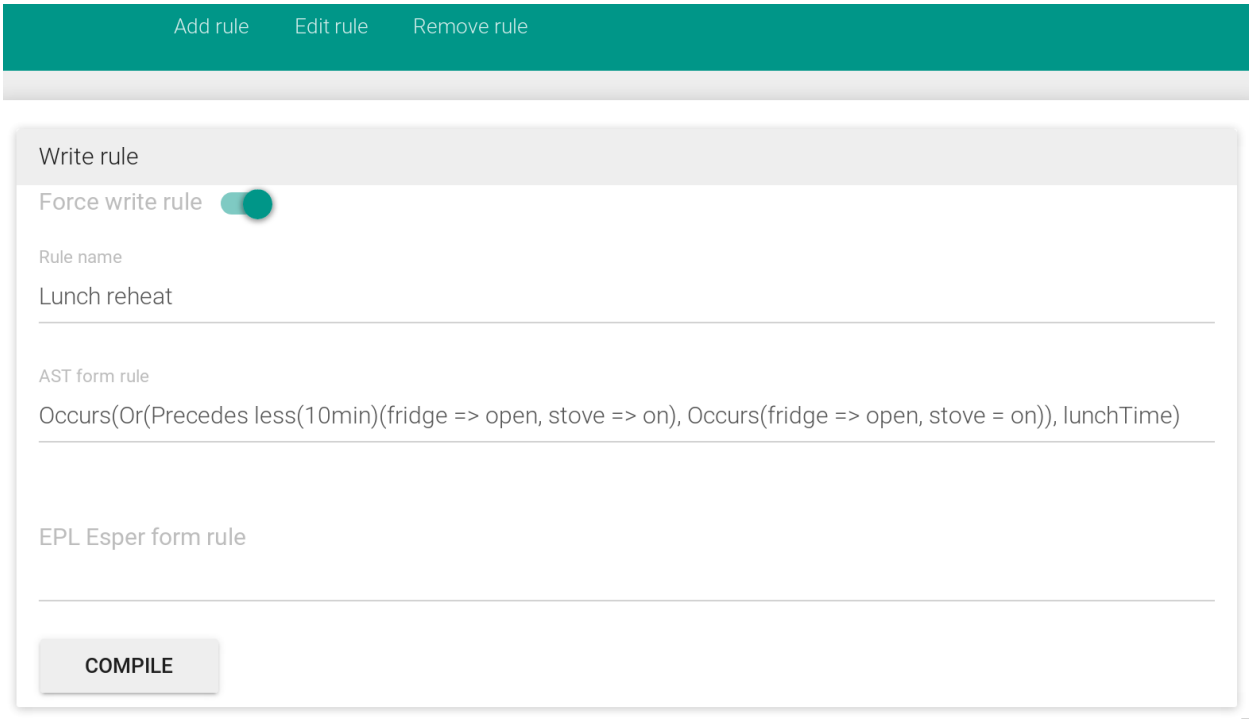


FIGURE 9: Interface d'administration de service.

Exécution du service

Pour être déployées, les règles compilées sont enregistrées auprès du moteur CEP. Notre moteur d'exécution de règle est basé sur Esper, un CEP open-source développé par EsperTech [1]. Esper propose des interfaces Java et .Net avec NEsper, en tant que bibliothèques pour développer des programmes événementiels. Nous avons choisi Esper parce qu'il s'agit d'un moteur CEP populaire utilisé à la fois dans l'industrie et dans la recherche. Esper fournit un langage spécifique et déclaratif pour le traitement d'évènements complexes, appelé EPL (pour Event Processing Language). Ce langage inclut tous les opérateurs de SQL, et ajoute des constructions telles que la définition et l'interaction avec des fenêtres, ainsi que pour la génération de sorties. EPL permet également de décrire des motifs d'évènements (ou "Patterns" EPL) à identifier dans un flux d'évènements temps réel, en utilisant des opérateurs pour ordonner les évènements, des contraintes temporelles, *etc.*. Ces opérateurs peuvent s'imbriquer et permettent de définir explicitement la politique de sélection d'évènements avec la clause "every". Esper permet de recevoir les données en sorties d'une règle, soit en mode push, par l'utilisation de listeners, soit en mode pull, en utilisant des itérateurs. Cependant, Esper ne permet pas de manipuler le concept d'état, d'où la nécessité d'encoder cette notion par des motifs purement évènementiels.

1. http://www.espertech.com/esper/

Forme canonique

La forme canonique des données produites par un domicile sensible au contexte permet de traiter les mesures d'interaction indépendamment de leur format original. Cette forme canonique de données, nommée *StreamEvent*, est ainsi une couche d'abstraction sur l'hétérogénéité des évènements des différentes sources, constituant un flux d'évènements uniforme. Dans cette représentation, chaque évènement est un quadruplet comprenant le type de l'évènement, sa localisation, sa valeur, et l'horodatage de son occurrence.

$$K$$

(pour Kinds) est l'ensemble des types d'interactions (*e.g.*, Freezer)

$$L$$

(pour Locations) est l'ensemble des localisation d'interactions (*e.g.*, Kitchen)

$$V$$

(pour Values) est l'ensembles des valeurs de chaque domaine d'interaction (*e.g.*, open/close)

$$\mathbb{N}$$

est l'ensemble des horodatages (timestamps) d'instances d'interactions

$$v \in \mathbb{V} \in \bigcup_{j=0}^{m} D_{K_i}$$

$$t \in \mathbb{N}$$

in

$$s \in S = \mathcal{P}(K \times L \times \mathbb{V} \times \mathbb{N})$$

$$s = \{(k_0, l_0, v_0^{D_{k_0}}, t_0), \ldots, (k_n, l_o, v_n^{D_{k_n}}, t_n)\}$$

Notons que le flux d'évènements en format StreamEvent ainsi défini est plus bas niveau que le format des rôle défini en Section 3.3. En effet, comme dans les langages de CEP, utilisés comme cible, les évènements sont toujours ponctuels (*i.e.*, n'ont pas de durée)[2], nous ne pouvons pas réutiliser directement les évènements de rôles, qui étaient estampillés par des intervalles de temps (les périodes). En revanche, ce nouveau format permet de détecter de manière plus précoce certains évènements complexes, sans attendre la fin d'un évènement, pouvant avoir une durée non négligeable. Par exemple, prenons la règle vérifiant si une porte reste ouverte pendant plus de 3 heures. Avec un log d'évènements de rôles, cette règle ne pouvait déclencher que lorsque la porte était refermée, car

2. Même dans les langages CEP avec une sémantique d'intervalle, les évènements élémentaires sont toujours ponctuels, comme discuté en Section 2.3.

c'est à ce moment-là que l'évènement "interaction de porte" se produisait. Au contraire, avec un log en format StreamEvent, il devient possible d'observer l'ouverture de porte indépendamment de sa fermeture, et donc de réagir, en principe, dès que le délai de 3 heures s'est écoulé. Nous verrons par la suite comment notre langage dédié atteint effectivement cette réactivité en s'appuyant sur le langage cible Esper. Ce niveau de réactivité n'était pas capital dans le contexte du Chapitre 3, où le signalement à la volée des dysfonctionnements des capteurs servait au service de maintenance, ayant un cycle d'intervention de l'ordre de plusieurs jours. Il en est tout autrement dans le contexte du chapitre actuel où l'on s'intéresse à une classe bien plus large de services, incluant des assistances de sécurité et des rappels d'activités.

Langage de règles

Le langage Maloya fournit une base d'expressions contraintes permettant d'exprimer l'ensemble des services nécessaires à l'assistance domiciliaire des personnes âgées. Ce langage permet aux intervenants de définir des services plus simplement que par des langages de programmations conventionnels. En effet, il restreint son expressivité aux seuls concepts nécessaires au domaine des services sensibles au contexte pour l'assistance domiciliaire. Pour ce faire, Maloya rend explicite les notions d'évènements, d'états et les opérateurs de composition.

Les intervenants peuvent définir les scénarios correspondant à un service à travers des règles constituées simplement par des compositions d'évènements et/ou d'états. En outre, à cause de la nature récurrente des services relatifs à l'assistance domiciliaire (*e.g.,* les activités quotidiennes telles que les repas ou la toilette), une règle est conçue pour que son exécution détecte chaque occurrence de la composition qu'elle décrit.

Il existe deux représentations du langage ; une représentation interne, qui constitue la représentation de base de notre langage, et une représentation textuelle qui permet d'exprimer des règles sous forme d'une phrase en langage naturel contraint. Il est important de noter que les représentations internes et textuelles de notre langage sont parfaitement équivalentes : la représentation textuelle étant simplement une formulation plus naturelle des expressions définies dans la représentation interne.

Évènements et États

Notre langage est construit autour de l'expression d'évènements et d'états. Un évènement est exprimé ainsi :

```
p becomes v
```

Un état est exprimé comme suit :

```
p is v
```

Dans chaque cas, `p` est l'identification d'une interaction (matérielle ou logicielle) et `v` appartient à l'ensemble des valeurs possibles pour l'interaction considérée. L'évènement `p becomes v` survient quand la mesure de l'interaction `p` signale une valeur `v`, si sa précédente valeur était différente. L'état `p is v` commence précisément quand l'évènement `p becomes v` se produit ; il se termine quand l'évènement `p becomes v'` se produit, où `v' ≠ v`. Nous voyons la période durant laquelle un état se maintient comme l'*intervalle de temps* d'un état.

La notion d'intervalle de temps est utilisée pour définir les opérateurs et leur sémantique. La Figure 10 présente la syntaxe et la sémantique informelle des évènements et états. Un évènement est représenté comme une pointe de signal. Un état est représenté par un signal rectangulaire ; ses points de départ et de fin correspondent à l'occurrence d'évènements provoquant son début et sa fin.

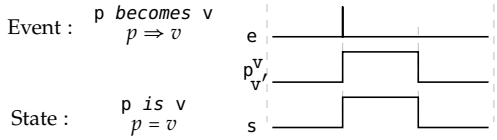


FIGURE 10: Semantique informelle et syntaxe des état et évènement.

Les états disposent également de filtres sur leur durée. Ces filtres permettent de borner la période d'un état avec une limite haute ou une limite basse. La limite haute de la durée d'un état est exprimée ainsi :

```
p is v less than t
```

et la limite basse est exprimée comme suit :

```
p is v for at least t
```

où `t` représente une période de temps.

Les opérateurs

Une *règle* dans notre langage est constituée d'opérateurs manipulant les états et/ou les évènements. Tous les opérateurs retournent des évènements. Plus précisément, un opérateur

produit un évènement de succès quand le contexte décrit par l'application de l'opérateur à des états/évènements est détecté. Puisqu'un opérateur retourne un évènement, les opérateurs qui prennent en argument un évènement peuvent prendre également une application d'opérateur comme argument. En revanche, les opérateurs qui prennent un état comme argument ne peuvent pas prendre une application d'opérateur comme argument. L'imbrication des opérateurs n'est ainsi pas arbitraire et suit notre étude de domaine.

La Figure 11 présente la double syntaxe des opérateurs de notre langage (représentation interne et textuelle), ainsi que sa sémantique informelle. La syntaxe est présentée sur la partie gauche. Sur la partie droite, une représentation graphique, suggère la sémantique de l'opérateur comme une fonction booléene définie sur le temps. Cette représentation permet de visualiser les évènements et les états impliqués dans une application d'opérateur, et la façon dont les opérateurs les combinent.

Les opérateurs définis dans le langage, listés dans la Figure 11, correspondent à des opérateurs de l'algèbre d'Allen sur les intervalles de temps [3].

3. James F. Allen [1983]. "Maintaining Knowledge About Temporal Intervals". In : *Commun. ACM* 26.11, p. 832–843.

Cependant, l'opérateur de l'algèbre d'Allen

```
X before Y
```

reconnaît toutes les occurrences de `X` qui précèdent toutes les occurrences de `Y`. Par exemple, dans la série d'intervalle suivante :

```
XXX YY
```

il identifie 6 paires (X,Y) correspondant à toutes les combinaisons des occurrences. Dans le cadre de la sensibilité au contexte pour l'assistance domiciliaire, il n'est pas nécessaire d'identifier plusieurs combinaisons des occurrences des évènements X et Y. Par exemple, si il y a eu une détection d'ouverture du frigidaire, suivie d'une détection de l'utilisation du four, il est inutile de prendre en considération les autre occurrences d'ouverture du frigidaire qui précèdent, ni les utilisations suivantes du four. Pour cette raison, notre opérateur Precedes prend en compte seulement la dernière occurrences de l'évènement X (ouverture du frigidaire dans l'exemple) immédiatement suivie par la première occurrence de l'évènement Y (utilisation du four dans l'exemple).

Nous avons ainsi généralisé les opérateurs d'Allen entre deux intervalles pour prendre en compte les multiples occurrences de ces intervalles. De plus, les opérateurs d'Allen n'acceptent que des intervalles non vides ; nous les avons donc généralisés, lorsque nécessaire, pour accepter aussi des évènements. En outre, pour certains opérateurs, nous avons défini

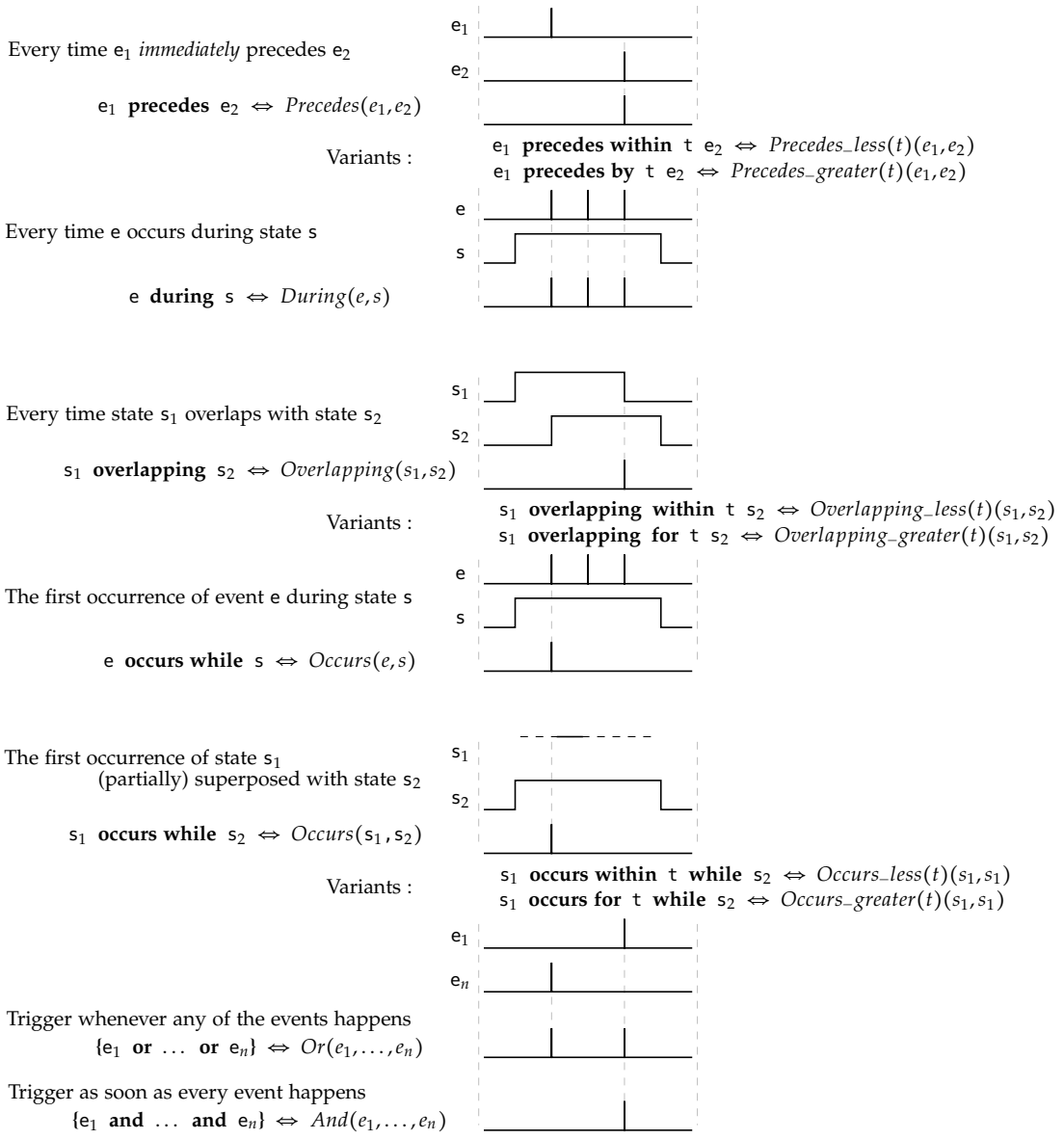


FIGURE 11: Sémantique informelle et syntaxe des opérateurs du langage Maloya.

une variante prenant en compte une contrainte temporelle, introduisant une limite basse et une limite haute pour la durée de la composition des opérandes.

Dans la suite du chapitre, par convention, les évènements seront notés avec la lettre e (possiblement indicée), et les états seront notées avec la lettre s (possiblement indicée).

Precedes

Cet opérateur rapporte un succès lorsqu'une occurrence de l'évènement e_1 précède *immédiatement* une occurrence de l'évènement e_2. Il ne doit pas y avoir d'autres occurrence des évènements e_1 ou e_2 entre les occurrences identifiées.

e_1 **precedes** e_2

Pour couvrir les scénarios d'assistance dans notre domaine cible, nous devons étendre l'expressivité de cet opérateur, avec des contraintes temporelles optionnelles. Plus précisément, nous introduisons deux variantes de cet opérateur :

e_1 **precedes** *within* t e_2
e_1 **precedes** *by* t e_2

La contrainte temporelle est définie par le paramètre `t`. Ces variantes définissent les limites hautes et basses des durées entre les occurrences des opérandes évènements.

During

Cet opérateur réussit chaque fois que l'évènement `e` se produit durant l'état `s`.

`e` **during** `s`

Il n'y a pas de versions de cet opérateur avec des contraintes temporelles.

Overlapping

Cet opérateur réussit quand une occurrence de l'état s_1 chevauche une occurrence suivante de l'état s_2. L'état s_1 commence avant le début de l'état s_2, et se termine durant s_2, comme montré en Figure 11.

s_1 **overlapping** s_2

Les versions avec contraintes temporelles définissent les limites hautes et basses de la durée de chevauchement des occurrences de ces états. Ces versions sont notées comme suit :

s_1 **overlapping** *within* t s_2
s_1 **overlapping** *for* t s_2

Occurs while

Cet opérateur est similaire à l'opérateur `during`, mais réussit *uniquement* pour la première occurrence de l'évènement `e` qui se produit durant l'état `s`.

`e` **occurs while** `s`

Une variante de cette règle est

s_1 **occurs while** s_2

Dans ce cas, l'opérateur réussit la première fois que l'état s_1 se superpose au moins partiellement avec s_2. Les versions avec contraintes temporelles de cet opérateur donnent les limites hautes et basses pour la durée de superposition des états. Elles sont notées comme suit :

s_1 **occurs** *within* t **while** s_2
s_1 **occurs** *for* t **while** s_2

Bien que les opérateurs d'Allen expriment un large éventail de situations, ils ne couvrent pas tous les besoins révélés par notre analyse de domaine. De nouveaux opérateurs doivent être introduits.

Or

En particulier, une disjonction d'événements est nécessaire pour permettre l'expression de contextes alternatifs. Une règle disjonctive est de la forme :

{ e_1 **or** ... **or** e_n **}**

Elle réussit quand le premier des évènements e_i se produit dans le flux.

And

Nous introduisons également une règle de conjonction de la forme :

{ e_1 **and** ... **and** e_n **}**

Cette règle réussit une fois que chacun des évènements e_i s'est produit au moins une fois.

Les opérandes

Comme expliqué précédemment, les opérandes sont soit des états, soit des évènements. Si les opérandes ne sont pas des compositions de règles (possible si l'opérande requise par l'opérateur est un évènement), ils sont alors l'expression des mesures d'interactions de l'utilisateur avec son environnement. Dans ce cas ils sont exprimé de façon à correspondre aux besoins du domaine, et plus particulièrement d'une installation.

Les interactions sont décrites dans une tables statique. Cette table définit les attributs de chaque interaction dans un domicile donné, comme illustré en Figure 12.

```
"name":{
  "location":"location",
  "kind":"kind",
  "values":["val1","val2"]
}
```

FIGURE 12: Table statique générique.

Cette table décrit le nom de l'interaction (*e.g.,* freezer), sa localisation (*e.g.,* kitchen), son type (*e.g.,* Freezer), et ses valeurs possibles (*e.g.,* open/close). Cependant, certaines interactions, principalement les interactions de présence, n'ont pas de localisation spécifiée dans cette table. En effet, ces interactions, étant possibles à de multiples endroits dans le domicile, sa localisation doit être précisée lors de son instanciation dans la règle entre parenthèse (*e.g.,Presence(Kitchen)*). De plus, il est possible de vouloir définir un opérande qui concerne toutes les interactions, par exemple pour définir une règle qui identifie les problèmes de communication de tout capteur. Pour ce faire nous avons introduit le mot clé *"Any"* :

```
Commfailure( Any ) becomes true
```

Exprimer un service

Nous illustrons maintenant l'utilisation de nos opérateurs en écrivant la règle en Maloya pour l'activité "Lunch Reheat", décrite précédemment (Section 4.2).

```
{ ( freezer becomes open precedes
    within 10 minutes stove becomes on )
  or
  ( freezer becomes open occurs while stove is on )
} occurs while lunchTime
```

FIGURE 13: Code Maloya textuel pour le service "Lunch Reheat".

Notons que cette spécification encode deux variantes d'un scénario : (1) prendre un repas depuis le frigidaire, ensuite allumer le four dans les 10 minutes qui suivent ; (2) prendre un repas depuis le frigidaire pour le mettre dans le four, qui est déjà allumé. Le tout durant la période du déjeuner.

Étapes de compilation

La compilation est faite en trois étape. La règle en langage textuel est compilée vers le langage interne, qui sont équivalents. La règle en langage interne est ensuite compilée vers une représentation intermédiaire qui est un pseudo-code EPL. Ce dernier est finalement compilé vers sa représentation finale en EPL.

Représentation interne

La première étape consiste à retranscrire le texte de la règle en langage dédié vers la représentation interne du langage Maloya ; cette traduction est définie par une correspondance biunivoque. Par exemple, dans le langage interne, la représentation de l'opérateur `e` **during** `s` devient $During(e,s)$. Ainsi, la traduction de notre exemple vers le langage interne de Maloya correspond à ce qui suit.

```
Occurs(Or(
    Precedes_less(10min)(freezer => open, stove => on),
    Occurs(freezer => open, stove = on)),
  lunchTime)
```

FIGURE 14: Code Maloya interne pour le service "Lunch Reheat".

où "=>" désigne un évènement qui se produit et "=" désigne un état qui se maintient.

Pseudo-code EPL

L'étape de compilation suivante génère du pseudo-code EPL. Ce pseudo-code utilise seulement les opérateurs EPL, mais n'instancie pas les attributs des évènements ; cette instanciation est faite ultérieurement. Cette étape implique plusieurs transformations. Premièrement, comme EPL n'offre pas la notion d'état, chaque état dans une règle est traduit par une séquence correspondant aux évènements qui marquent le début et la fin de l'état, ordonnée avec des opérateurs EPL standards. De cette façon, l'état exprimant le four comme étant allumé est traduit en une séquence EPL correspondant au four devenant allumé, suivi par n'importe quel évènement d'intérêt mais sans que le four n'ait été éteint. Par conséquent, l'opérateur "*Occurs(..., stove = on)*" est traduit en EPL par

```
1 stove => on -> ... and not stove => off
```

utilisant les opérateurs EPL "*and*", "*not*" et "->", qui correspond à *followed by* (suivi de la première occurrence de). De plus, dans cette phase de compilation, les contraintes temporelles sont traduites par l'utilisation explicite de la construction EPL "*where timer*:*within*" pour exprimer la limite haute, et l'utilisation explicite de "*and timer*:*interval*" pour exprimer la limite temporelle basse.

Par défaut, EPL limite la reconnaissance d'un évènement à une seule occurrence. Cette étape introduit donc la construction EPL "*every*", pour (1) gérer les occurrences d'un évènement en fonction de l'expressivité requise par l'opérateur, et

(2) pour gérer la nature récurrente des règles en elles-mêmes.

La composition d'opérateurs peut nécessiter la construction d'une fenêtre ("*window*"). L'opérateur utilisé comme opérande est alors compilé séparément et l'évènement produit est capturé dans une fenêtre nouvellement créée. L'opérateur composé peut alors utiliser cet évènement de la même manière qu'un évènement élémentaire, à travers l'identifiant de la fenêtre.

En poursuivant notre exemple, le résultat de cette phase de compilation en pseudo-code EPL est :

```
1  //Window1:
2  ( every freezer => open ->
3      stove => on
4      and not ( freezer => open ) where timer:within(10min) )
5  or
6  ( every stove => on ->
7      freezer => open
8      and not ( stove => off ) )

10 //Main rule:
11 every lunchTime => begin ->
12   Window1( timestamp > (lunchTime => begin).timestamp )
13   and not ( lunchTime => end )
```

FIGURE 15: Pseudo-code EPL pour le service "Lunch Reheat".

EPL Esper

L'étape finale consiste à obtenir la représentation EPL Esper, depuis le pseudo-code EPL, en effectuant l'instanciation des attributs nécessaires dans le flux d'événements canoniques (*i.e.*, la forme StreamEvent) en utilisant la table statique mentionnée précédemment.

De plus, cette étape lie dans la formule EPL tous les évènements comme provenant d'un même domicile (avec la contrainte EPL "`user=`*X*`.user`").

Les résultats de ces transformations nous permettent d'obtenir la représentation finale de la règle en EPL Esper qui est exécutée par le moteur Esper :

```
create window Wind.std:unique(location,kind,user)
select * from StreamEvent

insert into Wind select arg from pattern [
  (every arg=StreamEvent( location='Kitchen',kind='Freezer',
                          status='open' ) ->
    StreamEvent( location='Kitchen',kind='Stove',
                 status='on',user=arg.user )
    and not ( StreamEvent( location='Kitchen',kind='Freezer',
                           status='open',user=arg.user ))
    where timer:within (10min) )
  or ( every arg=StreamEvent( location='Kitchen',kind='Stove',
                              status='on' ) ->
       StreamEvent( location='Kitchen',kind='Freezer',
                    status='open',user=arg.user )
       and not StreamEvent( location='Kitchen',kind='Stove',
                            status='off',user=arg.user ))
]

select Cal_L_b,arg from pattern [
  every Cal_L_b= StreamEvent( location='Lunch',kind='Calendar',
                              status!='end') ->
    arg=Wind(timestamp>Cal_L_b.timestamp,
             user=Cal_L_b.user)
    and not StreamEvent( location='Lunch',kind='Calendar',
                         status='end',user=Cal_L_b.user ))
]
```

FIGURE 16: Code EPL compilé final pour le service "Lunch Reheat".

Notons que, même si ces étapes de transformations peuvent paraître simples, elles impliquent certaines subtilités. Les détails seront présentés dans le chapitre suivant.

Validation

Nous avons validé l'expressivité de notre langage en redéfinissant des services existant de DomAssist, incluant des services d'assistance et des services liés à la maintenance. L'exactitude du code produit par notre compilateur a été validée en comparant les résultats de l'exécution de nos règles compilées avec ceux des services existants et déployés sur la plateforme. Enfin, l'efficacité de notre langage a été validée en mesurant certains critères de performance pertinents pour notre domaine d'application.

Expressivité

Pour valider l'expressivité de notre langage, nous avons réimplanté 55 services déjà déployés dans DomAssist. Ces services sont les variations des 13 familles de règles listées en Figure 3. Des variations sont requises pour satisfaire les spécificités des différentes personnes. Par exemple, la détection des activités du quotidien comme la préparation du repas et le lever/coucher, doit être personnalisée pour chaque utilisateur en prenant en compte l'infrastructure déployée dans son domicile, les créneaux de reconnaissance des activités, et des périodes de temps entre les séquences d'interactions. Ainsi, pour préparer le petit déjeuner, une personne peut utiliser un presse agrume électrique et prendre un verre dans un placard spécifique, alors qu'une autre peut utiliser une bouilloire et ouvrir le frigidaire pour prendre du lait.

Réécrire un éventail de services nous permet de valider notre langage et ses concepts sous-jacents (évènements, états, opérateurs dérivés d'Allen), en vérifiant notamment qu'ils sont suffisamment expressifs pour couvrir des cas réels de services dans le domaine du maintien à domicile de personnes âgées.

Exactitude

Nous avons validé empiriquement l'exactitude de nos services compilés par inspection manuelle et des tests étendus. Nous avons inspecté manuellement les formes intermédiaires décrites précédemment (pseudo-code EPL, EPL Esper) pour les différentes règles, afin de nous assurer de la cohérence des résultats des différentes étapes de compilation.

Ensuite, nous avons validé la sortie du compilateur (*i.e.*, règle EPL résultante) en deux phases. Premièrement, nous avons testé les règles en les exécutant sur des fichiers de logs provenant du projet DomAssist. Ces logs contiennent des mesures d'évènements horodatés produits par tous les capteurs de l'infrastructure, qu'ils soient matériels ou logiciels. Cependant, ces logs ne contenaient pas les actions entreprises par les services existants. Pour compenser cette information manquante, nous avons implanté des scripts Perl implantant les spécifications de services originaux. Ces scripts sont plus simples à écrire que les vraies applications déployées puisqu'ils s'exécutent sur les logs de données : ils n'ont ainsi pas à traiter les résultats en temps réel, ni à se préoccuper de l'infrastructure de capteurs. Ainsi, les résultats des règles compilées ont été comparées automatiquement, dans cette première phase, avec ceux de scripts simulant les services d'origine. Ces comparaisons

nous ont permis de corriger certains schémas de compilation, jusqu'à trouver des résultats identiques.

Dans une seconde phase, nous avons connecté nos services au flux d'évènements temps réel sur la plate-forme de production pendant un mois, et évalué les résultats pour les services de neuf utilisateurs, en parallèle des services existants écrits en Java. Aucune différence n'a été observée entre les résultats des deux systèmes (Java et règles compilées depuis notre langage). De plus, cette phase nous a permis d'évaluer les performances à l'exécution de notre implantation.

Performances

Pour valider l'utilisabilité en pratique de notre approche à base d'un langage dédié, nous avons mesuré, en conditions réelles, les performances des règles EPL produites par notre compilateur avec deux indicateurs : la latence de détection, et la consommation mémoire de notre moteur d'exécution. Les règles sont exécutées sur le serveur de production du projet DomAssist (quad-core Intel(R) Xeon(R) E5-2407 v2 2.4 GHz CPU et 125GB RAM).

Latence La latence de détection d'une règle donne le temps entre le dernier évènement qui déclenche une règle, et le déclenchement effectif de la règle. Une latence faible indique que la règle est suffisamment réactive pour un usage pratique. D'après nos mesures, cette latence pour nos règles est toujours inférieure à une seconde. Cet ordre de grandeur est parfaitement compatible avec le type de règles qui sont implantées dans la plate-forme pour le maintien à domicile des personnes âgées, et convient y compris pour les services critiques de sécurité et de maintenance.

Occupation mémoire Nous avons également mesuré l'occupation mémoire de notre implantation pour nous assurer que le passage à l'échelle est possible pour des centaines d'utilisateurs et des dizaines de services. D'après nos mesures, la consommation mémoire est de 352MB en moyenne pour 55 règles, traitant les données de 129 domiciles pendant un mois 24 heures sur 24.

Encouragés par ces résultats, nous avons entrepris d'explorer la possibilité d'appliquer notre approche et notre implantation sur une architecture plus limitée, mais plus accessible et fréquemment utilisée en informatique ubiquitaire. Nous avons ainsi déployé notre implantation sur un Raspberry Pi 3 (quad-core ARM Cortex-A53 1.2 GHz CPU et 1GB RAM). Durant

cette expérimentation, nous avons exécuté les même 55 règles avec les mêmes 129 installations pendant un mois. La consommation mémoire constatée est de 173MB en moyenne. Cette moindre consommation mémoire sur le Raspberry Pi s'explique par son architecture 32-bit. Ces mesures nous montre que notre approche est applicable tant sur une infrastructure cloud que sur une infrastructure embarquée directement à l'intérieur du domicile.

Synthèse

Le langage Maloya permet d'exprimer des services sensibles au contexte en proposant une expressivité suffisante pour le domaine de l'assistance domiciliaire. Il permet de concevoir des règles en manipulant des données de contextes exprimées sous forme d'états et d'événements. Les opérateurs disponibles pour composer ces données offrent une abstraction suffisante pour masquer la complexité des règles de traitement événementiel. En effet, dans notre langage, les détails de bas niveau d'état et de gestion temporelle sont inclus dans la sémantique des opérateurs. Cette simplicité dans l'expression des règles est illustrée avec la règle pour l'activité "Lunch Reheat". Cette règle, en langage Maloya, est composée de trois évènements et deux états, et quatre opérateurs sont utilisés pour leurs composition. Une fois compilée en langage EPL, la règle contient cinq évènements, trois évènements *"not"*, trois opérateurs *"followed by"* et un opérateur *or*. Surtout, la règle Maloya est proche, dans sa structure, des besoins exprimés par les intervenants.

La redéfinition de services existants avec notre langage dédié et leur déploiement dans des conditions écologiques ont été un succès en terme d'efficacité de réalisation des tâches spécifiques aux intervenants. De plus notre implantation occupe des ressources matérielles satisfaisantes pour le domaine, montrant que notre approche est exploitable dans un environnement réel.

Name	Description / DSL	Metrics				Stakeholders
		DSL		EPL		
		#events	#states	#events	#not	
Presence dependency	Detect if cupboard status changes while no presence in kitchen `cupboard becomes open occurs while presence(Kitchen) is false`	1	1	2	1	Sensor installer
Departure alert	Detect if entrance door is opened at least for 5 minutes during calendar night time `door is open for 5 minutes occurs while nightTime`	1	1	2	2	Occupational therapist Caregiver
Door alert	Detect if entrance door is opened at least for 5 minutes during their is no presence in entrance `door is open occurs for 5 minutes while presence(Entrance) is false`	0	2	4	6	User Caregiver
Long inactivity	Detect if no movement in Bedroom since 24 hours `presence(Bedroom) is false for 24 hours`	0	1	1	1	Caregiver Occupational therapist
Fridge opened	Detect if fridge remains open at least 5 minutes `fridge is open for 5 minutes`	0	1	1	1	User Caregiver
Breakfast	Detect cupboard and coffeemaker opening (any order) during breakfast period `{cupboard becomes open and coffeeMaker becomes on} occurs while breakfastTime`	2	1	3	1	User Caregiver
Lunch reheat	Detect freezer opening and stove use in the 10 minutes following or freezer opening during stove use all during lunch period `{ ( freezer becomes open precedes within 10 minutes stove becomes on ) or ( freezer becomes open occurs while stove is on ) } occurs while lunchTime`	3	2	5	3	Caregiver User
Dinner	Detect fridge opening and microwave use (any order) during dinner period `{fridge becomes open and microwave becomes on} occurs while dinnerTime`	2	1	2	1	User Caregiver
Go to bed	Detect end of presence in bathroom and begin of presence in bedroom in the 10 minutes following during go-to-bed period `( presence(Bathroom) becomes false precedes within 10 minutes presence(Bedroom) becomes true ) occurs while gotobedTime`	2	1	3	2	Caregiver User
Wake-up	Detect end of presence in bedroom and begin of presence in kitchen in the 10 minutes following during go-to-bed period `( presence(Bedroom) becomes false precedes within 10 minutes presence(Kitchen) becomes true ) occurs while wakeupTime`	2	1	3	2	Caregiver User
Commfailure warning	Detect any sensor that fails to communicate `commfailure( Any ) becomes true`	1	0	1	0	Platform maintainer
Commfailure alert	Detect any sensor that has failed to communicate since 24 hours `commfailure( Any ) is true for 24 hours`	0	1	1	1	Sensor installer Platform maintainer
Battery alert	Detect battery level of any senser that become less than 5% `batteryLevel( Any ) becomes less than 5`	1	0	1	0	Sensor installer

TABLE 3: Exemple de services en langage Maloya textuel.

6
Compilateur

Nous avons présenté dans le chapitre précédent les étapes de compilation de manière succincte afin d'illustrer le résultat de la compilation globale d'une règle Maloya. Dans ce chapitre nous nous attachons à présenter les mécanismes de compilation détaillés et leur spécificités.

Sommaire

Aperçu

— Présentation de la sémantique des opérateurs EPL utilisés à l'issue de la compilation d'une règle en Maloya.

— Description de la sémantique des opérateurs Maloya et de leur schéma de compilation.

Notre langage repose sur des opérateurs couvrant le domaine des services sensibles au contexte. Ces opérateurs permettent de manipuler des concepts d'états et d'évènements et peuvent être composés afin de formuler des services couvrant les besoins des intervenants du domaine du maintien à domicile des personnes âgées. Nous présentons le schéma de compilation pour chacun des opérateurs ainsi que leur sémantique formelle.

Puisque nos opérateurs sont traduits dans des formules EPL, nous devons expliquer tout d'abord la sémantique d'EPL.

Sémantique informelle des opérateurs EPL

Le langage EPL fournit des opérateurs permettant de décrire des motifs à identifier dans un flux d'évènements. Cette section décrit les clauses fournies par EPL et utilisées dans la compilation finale des règles Maloya. Il n'existe pas, à notre connaissance, une sémantique formelle des opérateurs EPL ; nous décrivons donc de manière informelle la sémantique opérationnelle de chaque opérateur d'après le manuel de référence, enrichi par nos propres tests lorsque celui-ci n'est pas suffisamment détaillé.

La clause **pattern** définit l'ensemble des évènements à détecter, ainsi que leurs contraintes logiques et temporelles.

La clause **where** spécifie l'ensemble de contraintes associées à la règle. Il s'agit essentiellement de filtres et de conditions s'appliquant aux attributs des évènements.

La clause **timer :within**, utilisée à la suite de la clause "where", permet de terminer la sous-expression concernée. Cette clause teste si la sous-expression concernée devient vraie au bout d'une durée donnée. Dans le cas contraire, la sous expression sera considérée comme fausse. Cette clause nous permet de borner la durée d'un état par exemple.

L'expression **timer :interval** permet d'attendre une période de temps donnée avant de considérer cette expression comme vraie. Cette construction nous permet d'évaluer la persistance d'un état dans le temps par exemple.

La clause **every**, préfixant une expression, indique que l'expression doit redémarrer sitôt évaluée à vrai ou faux. En effet, avec le comportement par défaut d'un motif EPL, sans le mot-

clé "every", sitôt que la sous expression est évaluée à vrai ou faux, elle s'arrête.

La clause **followed-by**, noté "->", spécifie que l'expression à gauche de l'opérateur doit d'abord être vraie, et seulement après, l'expression de droite est évaluée pour trouver les évènements correspondants : `X -> Y`. Si l'expression `Y` ne comporte pas de clause **every**, comme expliqué ci-dessus, EPL ne reconnaîtra que sa première occurrence. Autrement dit, il ne peut pas y voir d'autres occurrences de `Y` entre les `X` et `Y` identifiés par cette règle.

La clause **not** inverse la valeur attendue pour qu'une expression réussisse. Les motifs d'expressions préfixés avec "*not*" sont considérés comme provisoirement vrais au départ, et deviennent définitivement faux quand l'expression incluse devient vraie. Dans les constructions EPL, cet opérateur est généralement utilisé en conjonction avec l'opérateur "*and*". Lors de notre compilation, nous utilisons toujours la construction `X and not Y`. Cette expression impose qu'il n'y ait pas d'évènements `Y` avant de reconnaître l'évènement `X`.

La clause **window**, associée à *create* permet de définir des fenêtres. Les fenêtres sont des constructions qui permettent de définir des évènements utilisateur, identifiables par un nom, et contenant un nombre quelconque d'occurrences.

La clause **insert into** insère des occurrences d'un évènement dans une fenêtre.

Sémantique et compilation des opérateurs Maloya

Dans la Section 5.4, nous avons introduit les opérateurs du langage Maloya et présenté de manière informelle leur comportement. Nous décrivons maintenant leur compilation en pseudo-code EPL, c'est-à-dire en séquence d'évènements, ainsi que leur sémantique formelle. La sémantique est présentée avant la règle de compilation de chaque opérateur et est formalisée sous forme d'une fonction booléenne du temps [1]. Nous notons $\langle\langle X \rangle\rangle$ la fonction booléenne donnant la sémantique d'une expression Maloya X. Cette fonction booléenne renvoie la valeur vrai exactement dans les moments où l'expression X doit réussir.

Pour un évènement élémentaire, la fonction $\langle\langle e \rangle\rangle$ renvoie vrai pour tous les moments où l'évènement e se produit. Pour un

1. Sharma CHAKRAVARTHY et al. [1994]. "Composite Events for Active Databases : Semantics, Contexts and Detection". In : *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., p. 606–617.

état, la fonction $\langle\!\langle s = v \rangle\!\rangle$ renvoie vrai à tous les moments où la dernière valeur fournie par le capteur s est v. La notation utilisée pour notre formalisation ne traitant que des évènements, nous exprimons les états en fonction de leurs évènements de début et de fin. Nous notons ainsi $[\mathrm{E}_{sb},\mathrm{E}_{se}]$ les évènements de début et de fin de l'état S.

Enfin, pour définir le schéma de compilation des opérateurs, nous notons $[\![X]\!]$, le code EPL produit pour une expression X du langage Maloya.

Notons que pour chaque évènement de base, c'est-à-dire les évènements d'interactions exprimés dans une règle Maloya, le code de cet évènement correspond à l'expression de cette interaction que nous notons $[\![e]\!] = e$. Les états étant décomposés en évènements à la compilation, ils ne sont pas concernés.

Precedes(e_1,e_2)

Precedes décrit la séquence de deux évènements E_1 et E_2. Precedes(E_1, E_2) est reconnu lorsqu'une occurrence de E_2 succède immédiatement à une occurrence de E_1. Ceci implique qu'il n'existe pas d'autres occurrences de E_1 ou de E_2 entre ces deux instants. Cet opérateur est défini ainsi :

$$\begin{aligned}\langle\!\langle \text{Precedes}(E_1, E_2) \rangle\!\rangle(t) = (\exists t_1)(\forall t_2)(&E_2(t) \wedge \\ &(t_1 < t) \wedge \\ &E_1(t_1) \wedge \\ &((t_1 < t_2 < t) \rightarrow\sim (E_1(t_2) \vee E_2(t_2))))\end{aligned}$$

La séquence d'évènements résultant de la compilation de l'opérateur *Precedes* spécifie qu'un évènement $\mathtt{e}_1$ doit être suivi par un évènement $\mathtt{e}_2$, en respectant la définition de l'opérateur Esper "followed-by", avec la condition supplémentaire qu'aucun autre évènement $\mathtt{e}_1$ ne doit se produire entre temps. La clause **every** permet de tester la règle pour chaque occurrence de $\mathtt{e}_1$. Notons que les fonctions auxiliaires de compilation telles que "WindowIfComplex" sont décrites dans la Section suivante.

$[\![$*Precedes*`(e1,e2)`$]\!]$=
  every `WindowIfComplex(e1)` →
    `WindowIfComplex(e2)`
      and not `WindowIfComplex(e1)`

During(e,s)

During filtre toutes les occurrences d'un évènement durant

la persistance d'un état donné. Plus précisément, soient E, E_{sb} et E_{se} trois évènements, où [E_{sb},E_{se}] décrit les évènements de début et de fin d'un état S. L'opérateur signale toutes les occurrences de E qui se produisent pendant l'intervalle démarré par E_{sb} et terminé par E_{se}. Plus précisément :

$$\langle\!\langle \text{During}(E,[E_{sb},E_{se}])\rangle\!\rangle(t) = (\exists t_1)(\forall t_2)(E(t)\wedge (t_1 < t)\wedge E_{sb}(t_1)\wedge ((t_1 < t_2 < t) \rightarrow\sim E_{se}(t_2)))$$

La compilation de l'opérateur *During* décrit une occurrence d'un évènement correspondant au début de l'état `s` (*i.e.,* la mesure d'interaction qui donne la valeur v_{sb}), suivie de toutes les occurrences de l'évènement `e`, sans qu'il n'y ait eu d'occurrence de l'évènement correspondant à la fin de l'état `s` (*i.e.,* la mesure d'interaction qui donne la valeur v_{se}). La clause *every* permet de tester la règle pour chaque occurrence de l'état `s`. La fonction "BoundedWindowIfComplex" garantit que si `e` est issue d'une séquence d'évènements fenêtrée, cette séquence à commencé après le début de l'état `s`.

```
⟦During(e,s)⟧=
  every Becomes(s,1) →
    every BoundedWindowIfComplex(e,Becomes(s,1))
    and not Becomes(s,0)
```

Occurs(e,s)

Occurs a une sémantique similaire à *During*, excepté qu'il reconnaît seulement la première occurrence d'un évènement durant la persistance d'un état donné. Soient E, E_{sb} et E_{se} trois évènements, où [E_{sb},E_{se}] décrit les évènements de début et de fin d'un état S. L'opérateur signale un évènement au moment de la première occurrence de E, qui se produit pendant l'intervalle démarré par E_{sb} et terminé par E_{se}. Plus précisément,

$$\langle\!\langle \text{Occurs}(E,[E_{sb},E_{se}])\rangle\!\rangle(t) = (\exists t_1)(\forall t_2)(E(t)\wedge (t_1 < t)\wedge E_{sb}(t_1)\wedge ((t_1 < t_2 < t) \rightarrow\sim (E_{se}(t_2)\vee E(t_2))))$$

La compilation de l'opérateur *Occurs* est semblable à celle de l'opérateur *During*, excepté que seule la première occurrence de l'évènement `e` déclenche un évènement.

⟦*Occurs*(e,s)⟧=
  every Becomes(s,1) →
    BoundedWindowIfComplex(e,Becomes(s,1))
    and not Becomes(s,0)

Occurs(s_1,s_2)

Cette version de l'opérateur *Occurs* exprime la superposition partielle de deux états persistants. Il s'agit d'une disjonction de deux évènements *Occurs(e,s)*. Soient $[E_{s_1b}, E_{s_1e}]$ les évènements de début et de fin d'un état S_1 et, $[E_{s_2b}, E_{s_2e}]$ les évènements de début et de fin d'un état S_2.

$$\langle\!\langle \text{Occurs}([E_{s_1b}, E_{s_1e}], [E_{s_2b}, E_{s_2e}]) \rangle\!\rangle(t) = \langle\!\langle Occurs(E_{s_2b}, E_{s_1b}, E_{s_1e}) \rangle\!\rangle(t) \vee \langle\!\langle Occurs(E_{s_1b}, E_{s_2b}, E_{s_2e}) \rangle\!\rangle(t)$$

La compilation de l'opérateur *Occurs* prenant deux états en argument définit une disjonction sur la version de *Occurs* qui reçoit un évènement et un état en argument. Pour que l'opérateur déclenche un évènement, l'état s_2 doit commencer durant l'état s_1 ou l'état s_1 doit commencer durant l'état s_2.

⟦*Occurs*(s_1,s_2)⟧=
  ⟦*Occurs*(Becomes(s_2,1),s_1)⟧
  or
  ⟦*Occurs*(Becomes(s_1,1),s_2)⟧

Overlapping(s_1,s_2)

Overlapping décrit le chevauchement de deux états S_1,S_2, où S_1 commence avant S_2. Soient $[E_{s_1b}, E_{s_1e}]$ les évènements de début et de fin d'un état S_1 et, $[E_{s_2b}, E_{s_2e}]$ les évènements de début et de fin d'un état S_2. Cet opérateur signale un évènement lorsqu'une occurrence de E_{s_1b} est suivie d'une occurrence de E_{s_2b}, elle-même suivie d'une occurrence de E_{s_1e}. De plus, il n'existe pas d'instant entre E_{s_1b} et E_{s_2b} pendant lequel se produit E_{s_1e}. Enfin, il n'existe pas d'instant entre E_{s_2b} et E_{s_1e} pendant lequel se produit E_{s_2e}. Plus spécifiquement,

$$\begin{aligned}\langle\!\langle \text{Overlapping}([E_{s_1b}, E_{s_1e}], [E_{s_2b}, E_{s_2e}]) \rangle\!\rangle(t) = \\ (\exists t_1)(\exists t_2)(\forall t_3)(E_{s_1e}(t) \wedge \\ (t_1 < t_2 < t) \wedge \\ E_{s_1b}(t_1) \wedge \\ E_{s_2b}(t_2) \wedge \\ ((t_1 < t_3 < t) \rightarrow\!\!\sim E_{s_1e}(t_3)) \wedge \\ ((t_2 < t_3 < t) \rightarrow\!\!\sim E_{s_2e}(t_3)))\end{aligned}$$

La séquence d'évènements de l'opérateur *Overlapping* reconnaît une occurrence de l'évènement de début de s_1, suivie d'une occurrence de l'évènement de début de s_2, sans qu'il n'y ait eu d'occurrence de fin de s_1, suivie d'une occurrence de fin de s_1, sans qu'il n'y ait eu d'occurrence de fin de s_2.

$[\![$*Overlapping*(s1,s2)$]\!]$ =
  every Becomes(s1,1) →
    Becomes(s2,1)
    and not Becomes(s1,0) →
      Becomes(s1,0)
      and not Becomes(s2,0)

And(e0,...,en)

And exprime la conjonction d'un ensemble d'évènements $E_1,\ldots,E_n$. L'opérateur signale un évènement la première fois que tous les évènements E_i se sont produits. Cela implique que le denier évènement manquant, E_{i_0}, vient de se produire pour la première fois. Cet opérateur est défini comme suit.

$$\begin{aligned}\langle\!\langle \mathrm{And}(E_1,\ldots,E_n)\rangle\!\rangle(t) = (\exists t_{i=1;n})(\exists i_0)(\forall t')((t_1 \le t) \wedge \cdots \wedge (t_n \le t) \wedge \\ E_1(t_1) \wedge \cdots \wedge E_n(t_n) \wedge \\ (t_{i_0} = t) \wedge \\ ((t' < t) \rightarrow\!\sim E_{i_0}(t')))\end{aligned}$$

Pour que l'opérateur `And` déclenche un évènement, il faut qu'une occurrence de chaque évènement qu'il prend en argument se soit produit.

$[\![$*And*(e1, ... ,en)$]\!]$=
  WindowIfComplex(e1) *and ... and* WindowIfComplex(en)

Or(e0,...,en)

Or exprime la disjonction d'un ensemble d'évènements $E_1,\ldots,E_n$. L'opérateur signale un évènement quand un évènement de cet ensemble se produit. La définition de cet opérateur est la suivante :

$$\langle\!\langle \mathrm{Or}(E_1,\ldots,E_n)\rangle\!\rangle(t) = E_1(t) \vee \cdots \vee E_n(t)$$

Or définit une disjonction normale et déclenche à chaque occurrence d'un évènement e_i.

```
⟦Or(e1, ... ,en)⟧=
  EveryIfLeaf(e1) or ... or EveryIfLeaf(en)
```

Fonctions internes au compilateur

La compilation des opérateurs vers du pseudo-code EPL utilise certaines fonctions auxiliaires de compilation. Par exemple il peut être nécessaire de vérifier si une fenêtre doit être créée pour une opérande, en fonction de son type ; ou encore d'extraire l'évènement de début ou de fin d'un état. Ces opérations conditionnelles sont factorisées dans les fonctions suivantes, internes au compilateur.

Notons que toutes ses fonctions acceptent en entrée une expression en représentation interne de Maloya et offrent en sortie une expression en pseudo-code EPL.

Becomes

Cette fonction traduit un état par l'évènement marquant son début ou sa fin, selon le deuxième argument booléen.

$Becomes(p = v, 1) \rightarrow p \Rightarrow v$
$Becomes(p = v, 0) \rightarrow p \Rightarrow v', v' \neq v$

FIGURE 17: Fonction Becomes. Traduit un état en un évènement.

WindowIfComplex

Cette fonction décide si une fenêtre doit être créée, en vérifiant si le fils du nœud courant est une séquence d'évènements. La fonction retourne l'identifiant de la fenêtre, si elle est crée ; sinon, le code du fils lui-même.

```
WindowIfComplex(e) →
  if e is leaf
  then
    ⟦e⟧
  else
    createWindow(e)
```

FIGURE 18: Fonction WindowIfComplex. Cette fonction vérifie si une fenêtre doit être créée.

BoundedWindowIfComplex

Cette fonction vérifie si le fils doit être fenêtré, de la même manière que la fonction WindowIfComplex. Si une fenêtre est

créée, un argument est ajouté à l'identifiant de la fenêtre afin d'assurer que l'évènement produit par celle-ci est bornée par le nœud passé en deuxième argument.

BoundedWindowIfComplex(e_1, e_2) →
```
  if e1 is leaf
  then
    [[e1]]
  else
    let window_id=createWindow(e1)
    window_id '(timestamp >' e2 '.timestamp)'
```

FIGURE 19: Fonction BoundedWindowIfComplex. Cette fonction vérifie si une fenêtre dont les évènements seront bornés doit être créée.

CreateWindow

Cette fonction calcule le code de l'évènement complexe `e` et le stocke dans un attribut de `e`. Elle retourne ensuite l'identifiant de la fenêtre nouvellement créée.

CreateWindow(e) →
```
  code_e ← [[e]];
  window_id_e
```

FIGURE 20: Fonction CreateWindow. Cette fonction calcule le code correspondant à l'évènement complexe qu'il a en argument et retourne l'identifiant de la fenêtre.

EveryIfLeaf

Cette fonction vérifie si le fils est une feuille et dans ce cas, retourne son code précédé de la clause *every* ; sinon, elle retourne simplement le code du fils.

EveryIfLeaf(e) →
```
  if e is leaf
  then
    'every' [[e]]
  else
    [[e]]
```

FIGURE 21: Fonction EveryIfLeaf. Cette fonction vérifie la clause *every* doit être ajoutée devant le code de l'évènement.

Compilation finale vers EPL

L'étape finale de compilation du pseudo-code EPL vers la représentation finale EPL peut survenir en deux circonstances : (1) une fois sur l'expression complète à compiler ; (2) pour chaque fenêtre créée par la fonction "CreateWindow", expliquée précédemment.

Ainsi, la construction *"select from pattern [[e]]"* est générée pour l'expression complète. De la même manière, pour chaque fenêtre "window_id$_e$", cette étape génère le code EPL implantant la fenêtre en utilisant les constructions EPL :

"create window window_id$_e$"
"insert into window_id$_e$"
"select from pattern [code$_e$]"

De cette façon, le "pattern" contient le code calculé par l'étape de compilation précédente.

Cette étape de compilation instancie également les évènements dans le pseudo-code EPL selon la forme *StreamEvent* en utilisant la fonction TranslateEvent 22. De plus, cette fonction lie tous les évènements dans la formule EPL à un même domicile, ou spécifie un domicile. Ce lien avec un domicile peut être fait de deux manières : (1) si un domicile particulier est spécifié dans la règle, alors l'identification du domicile est transmise à l'attribut "user" de chacun des évènements (*e.g.*, "user=userId") ; (2) si aucun domicile n'est spécifié, alors, pour lier toute la séquence à un même domicile, chaque évènement de la séquence prend l'attribut "user" du premier évènement de la séquence (*e.g.*, "user=X.user", X correspondant à l'identifiant du premier évènement de la séquence).

TranslateEvent

Cette fonction traduit les évènements pseudo-code EPL en évènements structurés EPL à partir des informations présentes dans la tables statique 12.

TranslateEvent$(p \Rightarrow v, e) \rightarrow$

```
  p=StreamEvent(location=p.location,
                kind=p.kind,
                value='v',
                user=e'.user')
```

TranslateEvent$(p \Rightarrow v, userId) \rightarrow$

```
  p=StreamEvent(location=p.location,
                kind=p.kind,
                value='v',
                user=userId)
```

FIGURE 22: Fonction TranslateEvent. Cette fonction transforme les évènements en pseudo-code EPL vers des évènements structurés EPL.

Synthèse

La compilation d'une règle en langage Maloya s'effectue en deux étapes : depuis la représentation interne vers le pseudo-code EPL, et depuis le pseudo-code vers le code EPL final. La

première étape constitue l'étape principale de la compilation. Elle consiste à appliquer le schéma de compilation de chacun des opérateurs suivant leur sémantique afin de compiler les concepts d'états, d'évènements, et les contraintes temporelles en une séquence d'évènements. La compilation vers le code final EPL permet principalement d'instancier les évènements de cette séquence en respectant la forme canonique *StreamEvent* et de lier chacune de ces informations de contexte à un même domicile.

On constate que tous nos opérateurs peuvent, en principe, être programmés directement dans le langage EPL. Cependant, on constate également que les formules dans ce langage peuvent s'avérer complexes. Ainsi, la ré-implantation récurrente des motifs représentés par des opérateurs comme *Occurs* ou *Overlapping* peut être une source importante d'erreurs. Au contraire, la compilation automatique de nos opérateurs depuis le langage Maloya rend ces motifs prédictibles dans leur fonctionnement et simplifie la programmation.

7
Conclusion

Nous proposons une nouvelle approche pour développer des services sensibles au contexte et spécifique au domicile. Nous avons analysé une variété de couches logicielles existantes dédiées au traitement de données dans le domaine du maintien à domicile de personnes âgées. Cette analyse nous a permis d'identifier des concepts clés et des opérations spécifiques pour les traitements sensibles au contexte. Sur cette base, nous avons développé un langage dédié sensible au contexte et son architecture logicielle. Ces deux composants permettent de mettre en synergie les intervenants du domaine en leur fournissant une approche unifiée pour concevoir et développer des services. Notre approche offre des abstractions et des notations spécifiques au contexte, au sein d'un paradigme orienté et centré données. Nous avons validé notre approche en l'appliquant à une plate-forme d'assistance au maintien à domicile, déployée actuellement chez des personnes âgées. Plus particulièrement, nous avons utilisé notre langage dédié pour redéfinir des services existants sur la plate-forme. Ces nouveaux services ont été déployés et testés avec succès en terme d'efficacité de réalisation des tâches spécifiques aux intervenants : détection des activités du quotidien, détection des risques impliquant les utilisateurs, surveillance des défaillances des capteurs, *etc.*

Discussion

Notre langage permet de faire le pont entre des concepts de haut niveau propres au domaine des services sensibles au contexte pour de l'assistance domiciliaire et des mécanismes de gestion d'évènements de bas niveau. Il permet d'exprimer des règles concises et de simplifier leur développement, en encapsulant les détails de gestion d'évènements dans le compilateur. En effet, les applications Java actuellement déployées dans la plate-forme implantent explicitement des automates

temporels, qui identifient les séquences d'évènements correspondant à chaque règle de notre langage. Les contraintes temporelles sont explicitement traitées en utilisant un service de minuterie. Ce service produit des évènements de temporisation qui sont insérés dans le flux d'évènements produit par l'infrastructure de capteurs. Dans notre langage, ces détails de bas niveau d'état et de gestion temporelle sont exprimés par des abstractions de haut niveau. Ainsi, le rôle de ces minuteries explicites correspond aux paramètres des variantes de nos opérateurs avec contraintes temporelles. Tout ceci rend nos règles plus simples à écrire et plus précises.

La validation de notre langage montre que notre approche permet un passage à l'échelle. En effet, notre langage permet d'unifier la définition de services, de maintenance et d'assistance, pour un domicile sensible au contexte en offrant l'expressivité suffisante pour décrire les concepts d'états et d'évènements propres au domaine. Par cette unification, le langage crée une synergie entre les intervenants qui peuvent par exemple partager l'expression de leur expertise. De plus, en simplifiant le développement des différents services, notre approche facilite leur personnalisation ce qui répond à un besoin important dans l'assistance domiciliaire, caractérisée par des fortes variations entre les personnes et entre les domiciles. En outre, les performances de notre implantation sont suffisantes pour le domaine. Effectivement, la latence de détection d'une règle, qui est de l'ordre de la seconde, convient même pour les services critiques comme les services de sécurité de l'utilisateur. Par ailleurs, l'occupation mémoire est relativement faible, et stable sur une longue période d'utilisation avec les flux de plus d'une centaine de domiciles. Par conséquent, il est possible d'exécuter notre approche sur des architectures à ressources limitées. Notre solution devient donc propice à un déploiement dans le domaine de l'assistance à domicile.

La première brique de notre approche, introduisant un modèle d'infrastructure, adresse des besoins en terme de fiabilité des informations contextuelles. Ce domaine, peu couvert par la littérature est central pour les services sensibles au contexte. Notre modèle d'infrastructure permet ainsi d'assurer en continu la fiabilité des informations contextuelles alimentant les services, modèle lui-même implanté sous forme de services.

Dans une première version, ce modèle d'infrastructure de capteurs était implanté par des règles Prolog. Ce modèle d'infrastructure est maintenant décrit à travers des services en lan-

gage Maloya. De cette façon, la méthodologie permettant d'assurer le bon fonctionnement des services sensibles au contexte est implanté comme n'importe quel service sensible au contexte. Ceci démontre la généralité de notre approche. Si, précédemment, ces services exprimés en Prolog nécessitaient l'intervention d'un programmeur spécialisé, notre langage rend plus accessible l'expression de services. Au delà de sa représentation textuelle que nous avons présentée, ce langage est destiné à être étendu à d'autres représentations (*e.g.,* visuelle) comme nous l'évoquerons plus bas. Cette perspective ouvrira l'accès au développement de services à des non-informaticiens. D'autre part, la formulation des règles en Prolog nécessitait, pour pouvoir les exécuter, que l'intervalle complet de l'état d'une interaction soit connu, limitant la réactivité dans la reconnaissance des défaillances. Contrairement à la formulation en Prolog, Maloya permet à une règle de retourner un résultat dans la seconde qui suit l'évènement déclencheur.

Perspectives

Notre approche est un premier pas vers la simplification du développement d'applications sensibles au contexte dans le domaine du maintien à domicile des personnes âgées. Toutefois, elle présente un certain nombre de limitations.

Définition de services par les intervenants

La première limitation de notre approche, et la plus importante, est que le langage Maloya, bien que plus facile d'accès qu'un langage de programmation général (ne serait-ce que par le petit nombre d'opérateurs), reste un langage difficile d'accès pour un publique non-informaticien. Cependant, il constitue une base idéale pour construire des couches de plus haut niveau. La représentation textuelle de haut niveau présentée dans ce document illustre cette possibilité. Une évaluation est en cours pour déterminer la compréhension du langage par des utilisateurs non-informaticiens. Si les résultats sont encourageants, cette étude sera étendue à la conception de services. Au-delà des représentations textuelles, des langages visuels peuvent être explorés. C'est un travail qui ne peut se faire qu'en collaborant étroitement avec les intervenants qui pourrons exprimer leurs besoins en terme de représentation visuelle des informations et avec des designers qui seront plus à même de traduire ces besoins en constructions d'un futur langage. En outre, un tel travail devrait permettre aux intervenants non-informaticiens de mieux s'approprier la technologie,

favorisant ainsi son acceptation.

Définir des actions

Notre langage présente également une limitation par rapport à d'autres approches : il ne dispose pas de constructions pour effectuer des actions sur l'environnement lorsqu'un contexte est reconnu. À l'heure actuelle les actions doivent être programmées dans un langage de programmation générique. Il serait utile d'étendre notre analyse du domaine pour couvrir également la partie contrôle des applications, et de dériver les concepts et notations nécessaires pour effectuer des actions, dans un esprit similaire au travail effectué sur le langage évènementiel Dura [1].

1. Steffen Hausmann [2014]. *The Language Dura : A Declarative Event Query Language for Reactive Event Processing*

Extension à d'autres domaines

Notre champs d'application reste limité au domaine du maintien à domicile des personnes âgées. Notre langage a donc été conçu et testé dans ce cadre, avec un ensemble d'opérateurs de composition spécifiques à ce domaine. Malgré tout, notre langage interne permet une expression riche de combinaisons de ces opérateurs. Il pourrait donc être intéressant d'explorer à l'avenir son applicabilité à d'autres domaines de services sensibles au contexte.

Nuance dans la valeur retournée par une règle

Nos règles retournent des valeurs booléennes. Toutefois, les informations contextuelles peuvent être plus générales que simplement binaires. Par exemple, une activité telle que la préparation de repas peut être détectée de façon plus nuancée, avec un score d'exécution compris par exemple entre 0 et 1, pour prendre en compte d'éventuelles déviations de la routine de l'utilisateur. Pour le moment, notre langage nécessite que ces variations soient codées dans différentes règles, ce qui n'est pas toujours efficace. Dans le futur, il peut être intéressant de considérer l'extension de notre approche à des opérateurs qui retournent des valeurs non-booléennes.

Étendre les cibles de compilation

Enfin, notre compilateur reste limité au langage EPL utilisé par le moteur CEP Esper. De nombreux autres moteurs CEP ont été développés, tant à des fins de recherche qu'à des fins plus industrielles. De même, de nombreuses autres implantations basées sur le traitement de flux d'évènements existent,

notamment l'écosystème Apache avec Spark et Flink. Une extension intéressante à fournir à notre langage serait ainsi de proposer une compilation vers différents langages de traitement événementiel.

Appendices

Grammaire du langage Maloya

Représentation interne

⟨*Rule*⟩ ::= ⟨*Name*⟩ ‘`:`’ ⟨*Atom*⟩

⟨*Atom*⟩ ::= ⟨*Event*⟩
| ⟨*State*⟩

⟨*Event*⟩ ::= ⟨*Operator*⟩
| ⟨*Role*⟩ ‘`=>`’ ⟨*Status*⟩

⟨*Operator*⟩ ::= ⟨*Precedes*⟩
| ⟨*During*⟩
| ⟨*Overlapping*⟩
| ⟨*Occurs*⟩
| ⟨*And*⟩
| ⟨*Or*⟩

⟨*Precedes*⟩ ::= ‘`Precedes`’ [(‘`_greater`’ | ‘`_less`’) ‘`(`’ ⟨*Time*⟩ ‘`)`’] ‘`(`’ ⟨*Event*⟩ ‘`,`’ ⟨*Event*⟩ ‘`)`’

⟨*During*⟩ ::= ‘`During(`’ ⟨*Event*⟩ ‘`,`’ ⟨*State*⟩ ‘`)`’

⟨*Overlapping*⟩ ::= ‘`Overlapping`’ [(‘`_greater`’ | ‘`_less`’) ‘`(`’ ⟨*Time*⟩ ‘`)`’] ‘`(`’ ⟨*State*⟩ ‘`,`’ ⟨*State*⟩ ‘`)`’

⟨*Occurs*⟩ ::= ‘`Occurs(`’ ⟨*Event*⟩ ‘`,`’ ⟨*State*⟩ ‘`)`’
| ‘`Occurs`’ [(‘`_greater`’ | ‘`_less`’) ‘`(`’ ⟨*Time*⟩ ‘`)`’] ‘`(`’ ⟨*State*⟩ ‘`,`’ ⟨*State*⟩ ‘`)`’

⟨*And*⟩ ::= ‘`And(`’ ⟨*Event*⟩ ‘`,`’ ⟨*Event*⟩ [‘`,`’ ⟨*Event*⟩] ‘`)`’

⟨*Or*⟩ ::= ‘`Or(`’ ⟨*Event*⟩ ‘`,`’ ⟨*Event*⟩ [‘`,`’ ⟨*Event*⟩] ‘`)`’

⟨*State*⟩ ::= ⟨*Role*⟩ ‘`=`’ ⟨*Status*⟩
| ‘`(`’ ⟨*Simple_State*⟩ ‘`)_delta(`’ (‘`<`’ | ‘`>`’) ⟨*Time*⟩ ‘`)`’

⟨*Role*⟩ ::= ⟨*Identifier*⟩ [‘`(`’ ⟨*Identifier*⟩ ‘`)`’]

⟨*Time*⟩ ::= { ‘`0`’..‘`9`’ } ⟨*Time_unit*⟩

⟨*Time_unit*⟩ ::= ‘`second`’ [‘`s`’]
| ‘`minute`’ [‘`s`’]
| ‘`hour`’ [‘`s`’]

⟨*Status*⟩ ::= { ‘`a`’..‘`z`’ | ‘`0`’..‘`9`’ }
| (‘`<`’ | ‘`>`’) { ‘`0`’..‘`9`’ }

⟨*Identifier*⟩ ::= (‘`a`’..‘`z`’ | ‘`A`’..‘`Z`’) [{ (‘`a`’..‘`z`’ | ‘`A`’..‘`Z`’ | ‘`_`’ | ‘`0`’..‘`9`’) }]

Représentation textuelle

⟨*Rule*⟩ ::= ⟨*Name*⟩ ‘`:`’ ⟨*Atom*⟩

⟨*Atom*⟩ ::= ⟨*Event*⟩
| ⟨*State*⟩

⟨*Event*⟩ ::= ⟨*Operator*⟩
| ⟨*Role*⟩ ‘`becomes`’ ⟨*Status*⟩

⟨*Operator*⟩ ::= ⟨*Precedes*⟩
| ⟨*During*⟩
| ⟨*Overlapping*⟩
| ⟨*Occurs*⟩
| ⟨*And*⟩
| ⟨*Or*⟩

⟨*Precedes*⟩ ::= ⟨*Event*⟩ ⟨*PrecedesOperator*⟩ ⟨*Event*⟩

⟨*PrecedesOperator*⟩ ::= ‘`precedes`’ [(‘`within`’ ⟨*Time*⟩) | (‘`by`’ ⟨*Time*⟩)]

⟨*During*⟩ ::= ⟨*Event*⟩ ‘`during`’ ⟨*State*⟩

⟨*Overlapping*⟩ ::= ⟨*State*⟩ ⟨*OverlappingOperator*⟩ ⟨*State*⟩

⟨*OverlappingOperator*⟩ ::= ‘`overlapping`’ [(‘`winthin`’ ⟨*Time*⟩) | (‘`for`’ ⟨*Time*⟩)]

⟨*Occurs*⟩ ::= ⟨*Event*⟩ ‘`occurs while`’ ⟨*State*⟩
| ⟨*State*⟩ ‘`occurs`’ [(‘`within`’ | ‘`for`’) ⟨*Time*⟩] ‘`while`’ ⟨*State*⟩

⟨*And*⟩ ::= ⟨*Event*⟩ ‘`And`’ ⟨*Event*⟩ [⟨*And*⟩]

⟨*Or*⟩ ::= ⟨*Event*⟩ ‘`Or`’ ⟨*Event*⟩ [⟨*Or*⟩]

⟨*State*⟩ ::= ⟨*Role*⟩ ‘`is`’ ⟨*Status*⟩ [(‘`less than`’ ⟨*Time*⟩) | (‘`for least`’ ⟨*Time*⟩)]

⟨*Role*⟩ ::= ⟨*Identifier*⟩ [‘`(`’ ⟨*Identifier*⟩ ‘`)`’]

⟨*Time*⟩ ::= { ‘`0`’..‘`9`’ } ⟨*Time_unit*⟩

⟨*Time_unit*⟩ ::= ‘`second`’ [‘`s`’]
| ‘`minute`’ [‘`s`’]
| ‘`hour`’ [‘`s`’]

⟨*Status*⟩ ::= { ‘`a`’..‘`z`’ | ‘`0`’..‘`9`’ }
| (‘`less than`’ | ‘`greater than`’) { ‘`0`’..‘`9`’ }

⟨*Identifier*⟩ ::= (‘`a`’..‘`z`’ | ‘`A`’..‘`Z`’) [{ (‘`a`’..‘`z`’ | ‘`A`’..‘`Z`’ | ‘`_`’ | ‘`0`’..‘`9`’) }]

Bibliographie

Unai Alegre, Juan Carlos Augusto et Tony Clark (2016). "Engineering context-aware systems and applications : A survey". In : *Journal of Systems and Software* 117.Supplement C, p. 55–83. issn : 0164-1212. doi : 10.1016/j.jss.2016.02.010.

James F. Allen (1983). "Maintaining Knowledge About Temporal Intervals". In : *Commun. ACM* 26.11, p. 832–843. issn : 0001-0782. doi : 10.1145/182.358434.

Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic et Rudi Studer (2010). "A Rule-Based Language for Complex Event Processing and Reasoning". In : *Web Reasoning and Rule Systems : Fourth International Conference, RR 2010, Bressanone/Brixen, Italy, September 22-24, 2010. Proceedings*, p. 42–57. isbn : 978-3-642-15918-3. doi : 10.1007/978-3-642-15918-3_5.

Juan C. Augusto et Chris D. Nugent (2004). "The Use of Temporal Reasoning and Management of Complex Events in Smart Homes". In : *Proceedings of the 16th European Conference on Artificial Intelligence*. ECAI'04. Valencia, Spain : IOS Press, p. 778–782. isbn : 978-1-58603-452-8. url : http://dl.acm.org/citation.cfm?id=3000001.3000165.

Ron Baecker, Kate Sellen, Sarah Crosskey, Veronique Boscart et Barbara Barbosa Neves (2014). "Technology to Reduce Social Isolation and Loneliness". In : *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility*. ASSETS '14. Rochester, New York, USA : ACM, p. 27–34. isbn : 978-1-4503-2720-6. doi : 10.1145/2661334.2661375.

Francisco J. Ballesteros, Enrique Soriano, Katia Leal Algara et Gorka Guardiola Muzquiz (2006). "Plan B : An Operating System for Ubiquitous Computing Environments". In : *4th IEEE International Conference on Pervasive Computing and Communications (PerCom 2006), 13-17 March 2006, Pisa, Italy*. IEEE Computer Society, p. 126–135. isbn : 0-7695-2518-0. doi : 10.1109/PERCOM.2006.43.

Louise Barkhuus et Anind Dey (2003). "Is Context-Aware Computing Taking Control away from the User ? Three Levels of Interactivity Examined". In : *UbiComp 2003 : Ubiquitous Computing : 5th International Conference, Seattle, WA, USA, October 12-15, 2003. Proceedings*. Springer Berlin Heidelberg, p. 149–156. doi : 10.1007/978-3-540-39653-6_12.

Christine Bauer (2012). "A Comparison and Validation of 13 Context Meta-Models." In : *ECIS*, p. 17. url : https://aisel.aisnet.org/ecis2012/17.

Chris Beckmann, Sunny Consolvo et Anthony LaMarca (2004). "Some Assembly Required : Supporting End-User Sensor Installation in Domestic Ubiquitous Computing Environments". English. In : *UbiComp 2004 : Ubiquitous Computing*. Sous la dir. de Nigel Davies, ElizabethD. Mynatt et Itiro Siio. T. 3205. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 107–124. isbn : 978-3-540-22955-1. doi : 10.1007/978-3-540-30119-6_7.

Johan Bengtsson et Wang Yi (2004). "Timed Automata : Semantics, Algorithms and Tools". In : *Lectures on Concurrency and Petri Nets : Advances in Petri Nets*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 87–124. isbn : 978-3-540-27755-2. doi : 10.1007/978-3-540-27755-2_3.

Valérie Bergua, Jean Bouisson, Jean-François Dartigues, Joel Swendsen, Colette Fabrigoule, Karine Pérès et Pascale Barberger-Gateau (2013). "Restriction in instrumental activities of daily living in older persons : Association with preferences for routines and psychological vulnerability". In : *The International Journal of Aging and Human Development* 77.4, p. 309–329. doi : 10.2190/AG.77.4.c.

Benjamin Bertran, Julien Bruneau, Damien Cassou, Nicolas Loriant, Emilie Balland et Charles Consel (2014). "DiaSuite : A tool suite to develop Sense/Compute/Control applications". In : *Science of Computer Programming* 79, p. 39–51. doi : 10.1016/j.scico.2012.04.001.

Mike Botts, George Percivall, Carl Reed et John Davidson (2008). "GeoSensor Networks". In : sous la dir. de Silvia Nittel, Alexandros Labrinidis et Anthony Stefanidis. Berlin, Heidelberg : Springer-Verlag. Chap. OGC®Sensor Web Enablement : Overview and High Level Architecture, p. 175–190. isbn : 978-3-540-79995-5. doi : 10.1007/978-3-540-79996-2_10.

J. Bruneau, W. Jouve et C. Consel (2009). "DiaSim : A parameterized simulator for pervasive computing applications". In : *Mobile and Ubiquitous Systems : Networking Services, MobiQuitous, 2009. MobiQuitous '09. 6th Annual International*, p. 1–10. doi : 10.4108/ICST.MOBIQUITOUS2009.6851.

AJ Brush, Bongshin Lee, Ratul Mahajan, Sharad Agarwal, Stefan Saroiu et Colin Dixon (2011). "Home automation in the wild : challenges and opportunities". In : *proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, p. 2115–2124. DOI : 10.1145/1978942.1979249.

Loïc Caroux, Charles Consel, Lucile Dupuy et Hélène Sauzéon (2014). "Verification of Daily Activities of Older Adults : A Simple, Non-Intrusive, Low-Cost Approach". In : *ASSETS - The 16th International ACM SIGACCESS Conference on Computers and Accessibility*. Rochester, NY, United States, p. 43–50. DOI : 10.1145/2661334.2661360.

A. Carteron, C. Consel et N. Volanschi (2016). "Improving the Reliability of Pervasive Computing Applications by Continuous Checking of Sensor Readings". In : *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, p. 41–50. DOI : 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0029.

Adrien Carteron, Charles Consel et Nic Volanschi (2018). "A Domain-Specific Approach to Unifying the Many Dimensions of Context-Aware Home Service Development". In : *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom) (PerCom 2018)*. Athens, Greece.

Damien Cassou, Julien Bruneau, Charles Consel et Emilie Balland (2012). "Toward a tool-based development methodology for pervasive computing applications". In : *IEEE Transactions on Software Engineering* 38.6, p. 1445–1463. DOI : 10.1109/TSE.2011.107.

Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar et S.-K. Kim (1994). "Composite Events for Active Databases : Semantics, Contexts and Detection". In : *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., p. 606–617. ISBN : 1-55860-153-8. URL : http://dl.acm.org/citation.cfm?id=645920.672994.

Marie Chan, Daniel Estève, Christophe Escriba et Eric Campo (2008). "A review of smart homes—Present state and future challenges". In : *Computer Methods and Programs in Biomedicine* 91.1, p. 55–81. ISSN : 0169-2607. DOI : http://dx.doi.org/10.1016/j.cmpb.2008.02.001.

L. Chen, J. Hoey, C. D. Nugent, D. J. Cook et Z. Yu (2012). "Sensor-Based Activity Recognition". In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Re-*

views) 42.6, p. 790–808. ISSN : 1094-6977. DOI : 10.1109/TSMCC.2012.2198883.

L. Chen, C. D. Nugent et H. Wang (2012). "A Knowledge-Driven Approach to Activity Recognition in Smart Homes". In : *IEEE Transactions on Knowledge and Data Engineering* 24.6, p. 961–974. ISSN : 1041-4347. DOI : 10.1109/TKDE.2011.51.

Shiva Chetan, Anand Ranganathan et R. Campbell (2005). "Towards fault tolerance pervasive computing". In : *IEEE Technology and Society Magazine* 24.1, p. 38–44. ISSN : 0278-0097. DOI : 10.1109/MTAS.2005.1407746.

Charles Consel, Lucile Dupuy et Hélène Sauzéon (2015). "A Unifying Notification System To Scale Up Assistive Services". In : *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*. ACM, p. 77–87.

– (2017). "HomeAssist : An Assisted Living Platform for Aging in Place Based on an Interdisciplinary Approach". In : *Proceedings of the 8th International Conference on Applied Human Factors and Ergonomics (AHFE 2017)*. Springer.

Diane J Cook, Aaron S Crandall, Brian L Thomas et Narayanan C Krishnan (2013). "CASAS : A smart home in a box". In : *Computer* 46.7, p. 62–69. DOI : 10.1109/MC.2012.328.

Joëlle Coutaz et James L Crowley (2016). "A first-person experience with end-user development for smart homes". In : *IEEE Pervasive Computing* 15.2, p. 26–39. DOI : 10.1109/MPRV.2016.24.

Joëlle Coutaz, James L Crowley, Simon Dobson et David Garlan (2005). "Context is key". In : *Communications of the ACM* 48.3, p. 49–53.

Joëlle Coutaz, Emeric Fontaine, Nadine Mandran et Alexandre Demeure (2010). "DisQo : A user needs analysis method for smart home". In : *Proceedings of the 6th Nordic Conference on Human-Computer Interaction : Extending Boundaries*. ACM, p. 615–618.

J. Criel, L. Claeys et L. Trappeniers (2011). "Deconstructing casensa : The caemp context-aware empowering platform". In : *Bell Labs Technical Journal* 16.1, p. 35–53. ISSN : 1089-7089. DOI : 10.1002/bltj.20484.

Gianpaolo Cugola et Alessandro Margara (2012). "Processing Flows of Information : From Data Stream to Complex Event Processing". In : *ACM Comput. Surv.* 44.3, 15 :1–15 :62. ISSN : 0360-0300. DOI : 10.1145/2187671.2187677.

Anind K. Dey (2001). "Understanding and Using Context". In : *Personal Ubiquitous Comput.* 5.1, p. 4–7. ISSN : 1617-4909. DOI : 10.1007/s007790170019.

Jeannette DURICK, Toni ROBERTSON, Margot BRERETON, Frank VETERE et Bjorn NANSEN (2013). "Dispelling Ageing Myths in Technology Design". In : *Proceedings of the 25th Australian Computer-Human Interaction Conference : Augmentation, Application, Innovation, Collaboration*. OzCHI '13. Adelaide, Australia : ACM, p. 467–476. ISBN : 978-1-4503-2525-7. DOI : 10.1145/2541016.2541040.

W.Keith EDWARDS et RebeccaE. GRINTER (2001). "At Home with Ubiquitous Computing : Seven Challenges". English. In : *Ubicomp 2001 : Ubiquitous Computing*. Sous la dir. de GregoryD. ABOWD, Barry BRUMITT et Steven SHAFER. T. 2201. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 256–272. ISBN : 978-3-540-42614-1. DOI : 10.1007/3-540-45427-6_22.

John FEMINELLA, Devika PISHAROTY et Kamin WHITEHOUSE (2014). "Piloteur : a lightweight platform for pilot studies of smart homes". In : *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM, p. 110–119. DOI : 10.1145/2676061.2674076.

Abdoulaye GAMATIÉ (2010). "Synchronous Programming : Overview". In : *Designing Embedded Systems with the SIGNAL Programming Language : Synchronous, Reactive Specification*. New York, NY : Springer New York, p. 21–39. ISBN : 978-1-4419-0941-1. DOI : 10.1007/978-1-4419-0941-1_2.

David GAY, Philip LEVIS, Robert von BEHREN, Matt WELSH, Eric BREWER et David CULLER (2003). "The nesC Language : A Holistic Approach to Networked Embedded Systems". In : *SIGPLAN Not.* 38.5, p. 1–11. ISSN : 0362-1340. DOI : 10.1145/780822.781133.

Ben GREENSTEIN, Eddie KOHLER et Deborah ESTRIN (2004). "A Sensor Network Application Construction Kit (SNACK)". In : *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*. SenSys '04. Baltimore, MD, USA : ACM, p. 69–80. ISBN : 1-58113-879-2. DOI : 10.1145/1031495.1031505.

Ramakrishna GUMMADI, Omprakash GNAWALI et Ramesh GOVINDAN (2005). "Macro-programming Wireless Sensor Networks Using Kairos". In : *Proceedings of the First IEEE International Conference on Distributed Computing in Sensor Systems*. DCOSS'05. Marina del Rey, CA : Springer-Verlag, p. 126–140. ISBN : 3-540-26422-1, 978-3-540-26422-4. DOI : 10.1007/11502593_12.

Steffen HAUSMANN (2014). *The Language Dura : A Declarative Event Query Language for Reactive Event Processing*.

S. HELMER et F. PERSIA (2016). "High-Level Surveillance Event Detection Using an Interval-Based Query Language". In :

2016 IEEE Tenth International Conference on Semantic Computing (ICSC), p. 39–46. DOI : 10.1109/ICSC.2016.19.

Karen Henricksen, Jadwiga Indulska et Andry Rakotonirainy (2002). "Modeling Context Information in Pervasive Computing Systems". English. In : *Pervasive Computing*. Sous la dir. de Friedemann Mattern et Mahmoud Naghshineh. T. 2414. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 167–180. ISBN : 978-3-540-44060-4. DOI : 10.1007/3-540-45866-2_14.

Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler et Kristofer Pister (2000). "System Architecture Directions for Networked Sensors". In : *SIGARCH Comput. Archit. News* 28.5, p. 93–104. ISSN : 0163-5964. DOI : 10.1145/378995.379006.

Dezhi Hong, Jorge Ortiz, Kamin Whitehouse et David Culler (2013). "Towards Automatic Spatial Verification of Sensor Placement in Buildings". In : *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*. BuildSys'13. Roma, Italy : ACM, 13 :1–13 :8. ISBN : 978-1-4503-2431-1. DOI : 10.1145/2528282.2528302.

E. Hoque, R.F. Dickerson, S.M. Preum, M. Hanson, A. Barth et J.A. Stankovic (2015). "Holmes : A Comprehensive Anomaly Detection System for Daily In-home Activities". In : *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on*, p. 40–51. DOI : 10.1109/DCOSS.2015.20.

Justin Huang et Maya Cakmak (2015). "Supporting Mental Model Accuracy in Trigger-action Programming". In : *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan : ACM, p. 215–225. ISBN : 978-1-4503-3574-4. DOI : 10.1145/2750858.2805830.

M. Jahn, M. Jentsch, C. R. Prause, F. Pramudianto, A. Al-Akkad et R. Reiners (2010). "The Energy Aware Smart Home". In : *2010 5th International Conference on Future Information Technology*, p. 1–8. DOI : 10.1109/FUTURETECH.2010.5482712.

Emil Jovanov, Aleksandar Milenkovic, Chris Otto et Piet C. de Groen (2005). "A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation". In : *Journal of NeuroEngineering and Rehabilitation* 2.1, p. 6. ISSN : 1743-0003. DOI : 10.1186/1743-0003-2-6.

Fahim Kawsar, Tatsuo Nakajima et Kaori Fujinami (2008). "Deploy spontaneously : Supporting end-users in building and enhancing a smart home". In : *UbiComp 2008 - Proceedings of the 10th International Conference on Ubiquitous Compu-*

ting. New York, NY, USA : ACM, p. 282–291. ISBN : 9781605581361. DOI : 10.1145/1409635.1409673.

Jeffrey KAYE (2017). "Making Pervasive Computing Technology Pervasive for Health & Wellness in Aging". In : *Public Policy & Aging Report*.

Jeffrey A KAYE, Shoshana A MAXWELL, Nora MATTEK, Tamara L HAYES, Hiroko DODGE, Misha PAVEL, Holly B JIMISON, Katherine WILD, Linda BOISE et Tracy A ZITZELBERGER (2011). "Intelligent systems for assessing aging changes : home-based, unobtrusive, and continuous assessment of aging". In : *Journals of Gerontology Series B : Psychological Sciences and Social Sciences* 66.suppl_1, p. i180–i190.

Narayanan C. KRISHNAN et Diane J. COOK (2014). "Activity recognition on streaming sensor data". In : *Pervasive and Mobile Computing* 10.Part B, p. 138–154. ISSN : 1574-1192. DOI : 10.1016/j.pmcj.2012.07.003.

Matthew L. LEE et Anind K. DEY (2015). "Sensor-based observations of daily living for aging in place". In : *Personal and Ubiquitous Computing* 19.1, p. 27–43. ISSN : 1617-4917. DOI : 10.1007/s00779-014-0810-3.

Ming LI, Murali MANI, Elke A. RUNDENSTEINER et Tao LIN (2011). "Complex event pattern detection over streams with interval-based temporal semantics". In : *Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems, DEBS 2011, New York, NY, USA, July 11-15, 2011*. Sous la dir. de David M. EYERS, Opher ETZION, Avigdor GAL, Stanley B. ZDONIK et Paul VINCENT. ACM, p. 291–302. ISBN : 978-1-4503-0423-8. DOI : 10.1145/2002259.2002297.

E.W.C. LIN (2004). *Software Sensors : Design and Implementation of a Programming Model and Middleware for Sensor Networks*. University of California, San Diego. URL : https://books.google.fr/books?id=sGo-AQAAIAAJ.

Dany LUSSIER-DESROCHERS, Hélène SAUZÉON, Charles CONSEL, Emilie BALLAND, Jennie ROUX, Yves LACHAPELLE, Valérie GODIN-TREMBLAY et Bernard N'KAOUA (2016). "Analysis of How People with Intellectual Disabilities Organize Information Using Computerized Guidance". In : *Disability and Rehabilitation : Assistive Technology*. To appear. DOI : 10.3109/17483107.2015.1136000.

Samuel MADDEN, Michael J. FRANKLIN, Joseph M. HELLERSTEIN et Wei HONG (2003). "The Design of an Acquisitional Query Processor for Sensor Networks". In : *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD '03. San Diego, California : ACM, p. 491–502. ISBN : 1-58113-634-X. DOI : 10.1145/872757.872817.

Sarah Mennicken, Jo Vermeulen et Elaine M. Huang (2014). "From Today's Augmented Houses to Tomorrow's Smart Homes : New Directions for Home Automation Research". In : *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington : ACM, p. 105–115. ISBN : 978-1-4503-2968-2. DOI : 10.1145/2632048.2636076.

Kazuya Murao, Haruka Mogari, Tsutomu Terada et Masahiko Tsukamoto (2013). "Evaluation Function of Sensor Position for Activity Recognition Considering Wearability". In : *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp '13 Adjunct. Zurich, Switzerland : ACM, p. 623–632. ISBN : 978-1-4503-2215-7. DOI : 10.1145/2494091.2495983.

Jürgen Nehmer, Martin Becker, Arthur Karshmer et Rosemarie Lamm (2006). "Living Assistance Systems : An Ambient Intelligence Approach". In : *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. Shanghai, China : ACM, p. 43–50. ISBN : 1-59593-375-1. DOI : 10.1145/1134285.1134293.

Pankesh Patel et Damien Cassou (2015). "Enabling high-level application development for the Internet of Things". In : *Journal of Systems and Software* 103, p. 62–84. ISSN : 0164-1212. DOI : 10.1016/j.jss.2015.01.027.

C. Perera, A. Zaslavsky, P. Christen et D. Georgakopoulos (2014). "Context Aware Computing for The Internet of Things : A Survey". In : *IEEE Communications Surveys Tutorials* 16.1, p. 414–454. ISSN : 1553-877X. DOI : 10.1109/SURV.2013.042313.00197.

M. Philipose, K.P. Fishkin, M. Perkowitz, D.J. Patterson, D. Fox, H. Kautz et D. Hahnel (2004). "Inferring activities from interactions with objects". In : *Pervasive Computing, IEEE* 3.4, p. 50–57. ISSN : 1536-1268. DOI : 10.1109/MPRV.2004.7.

Yongrui Qin, Quan Z. Sheng, Nickolas J.G. Falkner, Schahram Dustdar, Hua Wang et Athanasios V. Vasilakos (2016). "When things matter : A survey on data-centric internet of things". In : *Journal of Network and Computer Applications* 64.Supplement C, p. 137–153. ISSN : 1084-8045. DOI : 10.1016/j.jnca.2015.12.016.

A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell et M. D. Mickunas (2005). "Olympus : A High-Level Programming Model for Pervasive Computing Environments". In : *Third IEEE International Conference on Pervasive Computing and Communications*, p. 7–16. DOI : 10.1109/PERCOM.2005.26.

M. J. Rantz, M. Skubic, R. J. Koopman, L. Phillips, G. L. Alexander, S. J. Miller et R. D. Guevara (2011). "Using sensor networks to detect urinary tract infections in older adults". In : *2011 IEEE 13th International Conference on e-Health Networking, Applications and Services*, p. 142–149. DOI : 10.1109/HEALTH.2011.6026731.

Parisa Rashidi et Alex Mihailidis (2013). "A survey on ambient-assisted living tools for older adults". In : *IEEE journal of biomedical and health informatics* 17.3, p. 579–590. DOI : 10.1109/JBHI.2012.2234129.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman et Yasmin Kafai (2009). "Scratch : Programming for All". In : *Commun. ACM* 52.11, p. 60–67. ISSN : 0001-0782. DOI : 10.1145/1592761.1592779.

Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell et Klara Nahrstedt (2002). "A Middleware Infrastructure for Active Spaces". In : *Pervasive ComputingPervasive Computing* Middleware.October-November. DOI : 10.1109/MPRV.2002.1158281. published.

Neal Rosen, Rizwan Sattar, Robert W. Lindeman, Rahul Simha et Bhagirath Narahari (2004). "HomeOS : Context-Aware Home Connectivity". In : *Proceedings of the International Conference on Wireless Networks, ICWN '04, Volume 2 & Proceedings of the International Conference on Pervasive Computing and Communications, PCC'04, June 21-24, 2004, Las Vegas, Nevada, USA*. Sous la dir. d'Hamid R. Arabnia, Laurence Tianruo Yang et Chi-Hsiang Yeh. CSREA Press, p. 739–744. ISBN : 1-932415-39-4. URL : http://iihm.imag.fr/daassi/pourOlfa/contextSensitiveWidgets/Rosen_PCC_2004.pdf.

D. Sadoun, C. Dubois, Y. Ghamri-Doudane et B. Grau (2011). "An Ontology for the Conceptualization of an Intelligent Environment and Its Operation". In : *Artificial Intelligence (MICAI), 2011 10th Mexican International Conference on*, p. 16–22. DOI : 10.1109/MICAI.2011.32.

Amit Sheth, Cory Henson et Satya S. Sahoo (2008). "Semantic Sensor Web". In : *IEEE Internet Computing* 12.4, p. 78–83. DOI : 10.1109/MIC.2008.87.

J.A. Stankovic, Insup Lee, A. Mok et R. Rajkumar (2005). "Opportunities and obligations for physical computing systems". In : *Computer* 38.11, p. 23–31. ISSN : 0018-9162. DOI : 10.1109/MC.2005.386.

Ryo Sugihara et Rajesh K. Gupta (2008). "Programming Models for Sensor Networks : A Survey". In : *ACM Trans. Sen.*

Netw. 4.2, 8 :1–8 :29. ISSN : 1550-4859. DOI : 10.1145/1340771.1340774.

D. Surie, O. Laguionie et T. Pederson (2008). "Wireless sensor networking of everyday objects in a smart home environment". In : *Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on*, p. 189–194. DOI : 10.1109/ISSNIP.2008.4761985.

Mark Weiser (1993). "Some Computer Science Issues in Ubiquitous Computing". In : *Commun. ACM* 36.7, p. 75–84. ISSN : 0001-0782. DOI : 10.1145/159544.159617.

Kamin Whitehouse, Cory Sharp, Eric Brewer et David Culler (2004). "Hood : A Neighborhood Abstraction for Sensor Networks". In : *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*. MobiSys '04. Boston, MA, USA : ACM, p. 99–110. ISBN : 1-58113-793-1. DOI : 10.1145/990064.990079.

Kamin Whitehouse, Feng Zhao et Jie Liu (2006). "Wireless Sensor Networks : Third European Workshop, EWSN 2006, Zurich, Switzerland, February 13-15, 2006. Proceedings". In : Berlin, Heidelberg : Springer Berlin Heidelberg. Chap. Semantic Streams : A Framework for Composable Semantic Interpretation of Sensor Data, p. 5–20. ISBN : 978-3-540-32159-0. DOI : 10.1007/11669463_4.