Ordonnancement dans l'habitat intelligent

Alexandre Demeure, Sybille Caffiau

Revue Ouverte d'Intelligence Artificielle
Brouillon d'article, version du 25 mars 2024

Ordonnancement dans l'habitat intelligent

Alexandre Demeure, Sybille Caffiau[a]

[a] Univ. Grenoble Alpes, CNRS, Grenoble INP[(a)], LIG, 38000 Grenoble, France
E-mail : firstName.LastName@univ-grenoble-alpes.fr.

[(a)]Institute of Engineering Univ. Grenoble Alpes

Résumé. — Cet article aborde le problème de l'ordonnancement des actions portants sur les effecteurs d'un habitat intelligent. Le déclanchement de ces actions peut être demandé par les habitants ou des programmes gérant des automatismes. Nous montrons que ce problème est complexe et ne peut être résolu a priori car il dépend du contexte. Nous défendons l'idée que le problème doit être abordé sous l'angle d'un système d'exploitation dont le moteur d'ordonnancement pourrait être basé sur le langage CCBL (Cascading Contexts Based Language). CCBL est un langage de programmation pour utilisateur finaux de l'habitat. Il permet de coder des automatismes basés sur les appareils et services de l'habitat. Nous donnons plusieurs exemples de stratégies d'ordonnancement programmées avec CCBL et montrons que programmer de telles stratégies n'est pas fondamentalement différent que d'utiliser CCBL pour programmer des automatismes. Les compétences acquises sur l'une de ces tâches seront donc réutilisables sur l'autre.

Mots-clés. — Habitat Intelligent, DSL, CCBL, Programmation par l'utilisateur final, Developpement par l'utilisateur final, ordonnancement.

1. Introduction

L'habitat intelligent (aussi nommé habitat domotique ou encore smart home) fait l'objet de nombreux travaux tant dans le domaine académique que dans le domaine industriel. Comme l'ont noté Mennicken et al., derrière le terme d'habitat intelligent se cache des définitions sensiblement différentes [18]. Il peut être utilisé pour désigner la composition d'appareils contrôlables à distance (objets connectés) suivant alors un point de vue entièrement centré technologie. Il peut aussi qualifier un habitat s'adaptant aux besoins des habitants, suivant alors un point de vue centré usage. En pratique, les foyers actuellement équipés de solutions domotiques décrivent leur habitat comme augmenté par du contrôle à distance et des automatismes plutôt qu'intelligent [4, 10].

Les usages d'un habitat intelligent sont variés [2, 4, 7, 10, 17, 18] : économies d'énergie (chauffage principalement) ; sécurité (contre une intrusion ou pour sécuriser l'habitat vis-à-vis d'enfant) ; le confort (gestion automatique des lumières, faciliter d'accès aux services multimédias, etc.) ; gestion à distance (être rassuré sur ce qui se passe, fermer les volets si on a oublié, etc.) ; notifications (ex : sortir les poubelles, savoir quand la machine a terminé de laver le linge) ; ou tout simplement prendre plaisir à bricoler soit même dans la maison (pour les makers notamment, la satisfaction de faire par soi-même et d'avoir surmonté des défis techniques est une motivation au moins aussi importante que le but final). Notons aussi, comme l'ont souligné Coutaz et Crowley [7], que les idées d'usages viennent et varient au cours du temps en vivant avec le système.

Il existe deux grandes approches permettant aux habitants d'exprimer ce que doit faire l'habitat pour répondre à leurs besoins. La première approche consiste à commander directement les appareils et services (en utilisant différentes modalités d'interaction comme la voix, des télécommandes logicielles ou physiques). La seconde approche consiste à utiliser des programmes qui permettent de piloter les appareils et services en fonction des données observées par les capteurs (matériels et logiciels). Ces programmes peuvent être définis par l'habitant lui-même (comme proposé par la plupart des boxes domotiques), par des programmeurs professionnels (via des magasins d'application) ou par des systèmes d'apprentissages automatiques comme le Thermostat NEST[(1)]. Notons que le programme peut aussi piloter l'ensemble des appareils et services de l'habitat et apprenne à se comporter de manière à satisfaire l'utilisateur et à apprendre en fonction de feedbacks émis par l'utilisateur [20, 14].

Au sein de l'habitat coexistent donc plusieurs acteurs qui agissent sur les appareils et services : les acteurs humains (habitants, invités, ...) et les acteurs logiciels (les programmes). Dans certains contextes, différents acteurs peuvent vouloir appliquer des commandes différentes aux mêmes appareils ou services. Par exemple, un acteur humain et un des programmes qu'il a installés peuvent vouloir agir sur le chauffage ; l'humain pour augmenter la température (car il reçoit des invités à l'improviste) et le programme thermostat pour la baisser [25]. Le résultat de ces différences est un comportement erratique et insatisfaisant de l'habitat. La problématique que nous abordons

[(1)] https://nest.com/fr/thermostats/nest-learning-thermostat/overview/

dans cet article est celle de l'ordonnancement de l'accès aux commandes par ces différents acteurs. Nous proposons d'aborder ce problème sous l'angle du Développements par l'Utilisateur Final (DUF, ou EUD en anglais) (cf. [5]) et présentons comment le langage CCBL (Cascading Contexts Based Language) [21, 22] peut être utilisé pour exprimer la priorité entre ces acteurs selon le contexte.

Dans la section 2, nous décrivons les différents acteurs de l'habitat et explicitons le problème de l'ordonnancement entre eux. La section 3 décrit notre approche du problème qui s'articule autour du langage CCBL. Enfin nous concluons et donnons des perspectives à ce travail.

2. Les différents acteurs de l'habitat et leur ordonnancement

L'intérêt du contrôle des objets de l'habitat par commande directe ou via l'utilisation de programmes peut être évalué en fonction du contexte. Nous présentons dans les deux premières sections ces modalités de contrôle et les conditions pour lesquelles elles sont adaptées. Le problème de l'ordonnancement entre ces modalités est ensuite décrit et discuté dans le cadre de l'habitat intelligent.

2.1. La commande directe

La forme la plus directe pour un habitant de faire réagir l'habitat en fonction de ses besoins est d'utiliser une interface pour commander les appareils et services de l'habitat. Différentes solutions d'interactions sont proposées. La plus courante, offerte par toutes les boxes domotiques, est de contrôler directement les appareils et services à l'aide de télécommandes logicielles graphiques. Les assistants comme Google Home ou Amazon Alexa permettent quant à eux d'utiliser des commandes vocales. Enfin, il existe des solutions de télécommandes physiques dédiées, comme celle proposée par [23] ou par Sevenhugs[(2)], qui permettent de piloter les appareils en les pointant.

De nombreuses solutions permettent l'accès aux appareils au sein de l'habitat ou à distance. Le scénario 1 (Table 2.1) illustre l'utilisation de la commande directe via deux médias distincts : un bouton physique utilisé au sein de l'habitat et une application sur smartphone utilisée hors de l'habitat.

La commande directe n'est cependant pas la panacée. Elle peut se révéler fastidieuse lorsque de nombreux appareils et services doivent être commandés pour atteindre l'objectif de l'habitant. Par exemple, mettre la maison en « mode relaxant » peut impliquer d'opter pour une lumière tamisée, mettre une musique particulière avec un volume modéré, baisser les volets, etc. Dans ce dernier cas, la solution adoptée par les boxes domotiques est de laisser la possibilité de définir des scènes (Demeure et al., 2015), c'est-à-dire des macros ou configurations d'appareils et services qui peuvent être utilisés à la demande. Les scènes permettent ainsi de spécifier les actions à réaliser dans des programmes.

[(2)] `https://www.kickstarter.com/projects/901859853/sevenhugs-smart-remote-the-first-remote-for-everyt/`

Avant de partir au travail, Thomas remplie sa machine à laver le linge. En ajoutant la lessive, il s'aperçoit qu'il n'en a presque plus et décide d'en commander en appuyant sur le bouton Amazon Dash qu'il a collé sur le fronton de sa machine. Sa machine est prête à démarrer mais il préfère la mettre en attente avant de partir au travail afin de ne pas laisser le linge mouillé jusqu'au soir. Au bureau, avant de rentrer, il utilise la télécommande logicielle de son smartphone pour démarrer la machine à laver, de sorte qu'elle termine à peu près au moment où il sera rentré.

TABLE 2.1. Scénario 1. Thomas gère son linge.

2.2. LES PROGRAMMES

Les programmes déployés dans l'habitat visent à automatiser certaines tâches. Cela permet d'alléger la charge mentale des habitants qui ont la charge de ces tâches habituellement. D'après la littérature [4, 7, 10, 18], les habitants peuvent déployer des automatismes dans plusieurs buts :

- augmenter la sécurité : verrouiller automatiquement portes, fenêtres et volets ou sécuriser l'accès à des parties dangereuses de l'habitat (en bloquant l'accès à la piscine extérieur pour les enfants par exemple) ;
- économiser l'énergie : gérer automatiquement le chauffage de sorte à moins consommer en fonction de la présence ou de l'absence des habitants ;
- faciliter la réalisation des tâches quotidiennes : allumer ou éteindre la lumière automatiquement en fonction de la luminosité extérieure et de la présence d'une personne dans une pièce ; rythmer la routine quotidienne des habitants en rappelant l'heure du départ pour l'école ou le jour de sortie des poubelles ;
- augmenter le sentiment de contrôle en cas d'absence : avoir la possibilité de consulter les valeurs des capteurs à distance (est ce que les volets sont bien fermés ?) ou être prévenu automatiquement que les enfants sont bien rentrés de l'école.

Les buts listés ci-dessus sont les plus courants mais, comme le notent [7], lorsqu'un habitat est équipé cela donne au fil du temps de nouvelles idées d'automatismes aux habitants. Ainsi de nouveaux programmes peuvent être ajoutés à l'initiative des habitants. Ces programmes peuvent être réalisés par les habitants eux-mêmes ou bien avoir été réalisés par des professionnels puis téléchargés et installés par les habitants.

Les programmes réalisés par des professionnels offrent un haut niveau d'abstraction aux habitants et permettent de mettre en œuvre des automatismes complexes. Le Thermostat NEST en est un exemple. Il est basé sur des algorithmes d'apprentissages automatiques que des non-programmeurs ne pourraient pas programmer eux-mêmes. Même si les résultats ne sont pas toujours à la hauteur des attentes [25], il est certain que ce type de programmes est nécessaire pour aborder des problèmes complexes comme l'optimisation de la consommation d'énergie dans l'habitat (et à plus forte raison si cela doit être coordonné à l'échelle d'un quartier ou une ville).

Les assistants comme Google Home [3] ou Amazon Alexa [4] offrent eux aussi des programmes, quoique de manière plus cachée. Ces programmes analysent les phrases reconnues par le système et sont activés si un schéma particulier est reconnu (par exemple un schéma pour reconnaitre les phrases du type : « Alexa, allume les lumières du salon »).

Certaines boxes domotiques permettent d'utiliser des programmes développés par des professionnels et conçus pour s'intégrer dans l'environnement de la boxe [10]. Ces programmes sont conçus pour être facilement configurables par les utilisateurs. Ils implémentent des automatismes courants comme par exemple le contrôle du thermostat (avec gestion de l'hystérésis). Les habitants paramètrent ensuite le programme pour l'adapter aux besoins du foyer (par exemple en réglant la température et les seuils d'hystérésis).

Enfin, pratiquement toutes les solutions domotiques offrent la possibilité aux habitants de programmer eux-mêmes leurs automatismes [10]. Les langages de programmations proposés par ces boxes sont très majoritairement dérivés du paradigme ECA (Event Conditions Actions). Les habitants définissent ainsi un ensemble de règles ECA qui peuvent être organisées en programmes, comme dans le cas du système AppsGate [8], pour coder les automatismes qu'ils souhaitent voir appliquer dans leur habitat.

Les programmes permettent donc à l'habitat d'agir de façon autonome, déchargeant ainsi les habitants de certaines tâches. Cependant, comme le remarque [9], les programmes sont basés sur une logique de procédure (peu flexible, gérant mal les exceptions) alors que les habitants fonctionnent plutôt par routines (les exceptions sont courantes, la routine doit être très flexible). Bien que la mise en œuvre de routines et non de procédures soit encore un problème ouvert, ce n'est qu'en couplant les programmes à des commandes directes qu'on peut se rapprocher de la flexibilité requise du point de vue de l'habitant.

2.3. Le problème de l'ordonnancement dans l'habitat intelligent

Dans un habitat dit intelligent, l'état des appareils et services de l'habitat est susceptibles d'être modifié par des programmes et/ou par les habitants. Il est inévitable que ces différents acteurs expriment des buts différents dans certains contextes. Le problème de l'ordonnancement dans l'habitat intelligent est de définir comment ces buts doivent être interprétés par le système : est-ce que des acteurs doivent avoir la priorité sur les autres ? Dans quels contextes ou concernant quels appareils ou services ? Dans les solutions actuelles le but suivi par le système est celui du dernier acteur à avoir donné un ordre mais est-ce toujours ce qu'il convient de faire ? Nous discutons dans cette section de la signification que peut avoir l'expression de buts différents et des stratégies possibles pour que le système gère cela.

Le cas le plus évident de buts divergents est celui de plusieurs habitants exprimant des besoins contradictoires par des commandes directes. On peut imaginer que les buts

[3] `https://store.google.com/fr/product/google_home`
[4] `https://www.amazon.fr/dp/B079PNT5TK/ref=fs_rad`

soient exprimés simultanément mais plus probablement ils le seront successivement, avec un délai pouvant être de quelques secondes ou de quelques heures (voir plus). Par exemple, un des habitant peut vouloir augmenter la température de la pièce alors qu'un autre peut vouloir la refroidir. Ce genre de divergence se règle actuellement par la discussion entre les habitants, le système ne prenant pas partie. Il est cependant tout à fait envisageable que le système puisse aider à arbitrer en fonction du contexte. Par exemple, les ordres provenant d'un enfant pourraient n'être appliqués que s'ils ne contredisent pas ceux d'un adulte. De façon plus fine, on peut envisager que certains appareils ou services ne soient accessibles que pour certains utilisateurs dans certains contextes (par exemple le contrôle de la télévision accessible seulement aux parents à certaines heures).

Tout comme les habitants, les programmes peuvent avoir des buts divergents. Le fait que chaque programme puisse modifier l'état des appareils et services indépendamment des autres peut être la cause de comportements indésirables. Par exemple, si deux programmes gèrent les portes et les fenêtres, l'un dans le but d'assurer la sécurité et l'autre dans le but d'assurer des économies d'énergie, alors il est possible que la sécurité de l'habitat soit compromise. Dans pareille situation, le programme assurant la sécurité devrait être prioritaire. Cela signifie que lorsque le programme de sécurité est activé, les changements d'états qu'il applique aux portes et aux fenêtres ne peuvent dès lors pas être modifiés par les autres programmes. Cela est si difficile à réaliser avec les systèmes actuels qu'en pratique nous avons constaté lors d'études de terrain que pour assurer cette propriété du système, l'usage était d'avoir deux sous-systèmes étanches l'un à l'autre (Demeure et al., 2015).

Le dernier type de divergence à prendre en compte est le cas où un habitant exprime par commande directe un ordre divergent par rapport à un programme. Dans les systèmes actuels la commande sera exécutée par le système, cependant cela n'est pas adapté à toutes les situations. Nous discutons ci-après des réponses que pourrait donner le système en les illustrant par des contextes qui nous semblent pertinents :

- La commande pourrait être ignorée ; cela peut être pertinent dans le cas où la commande est émise par un utilisateur disposant de moins de « droits » que celui ayant mis en place le programme (par exemple un enfant demandant à allumer la télévision du salon alors que les parents ont mis en place une règle l'interdisant). La contradiction peut aussi être issue de deux programmes. Par exemple un programme d'économie d'énergie peut chercher à éteindre les prises quand personne n'est à la maison alors qu'un programme de sécurité peut stipuler que certaines doivent rester allumées (par exemple la prise sur laquelle est branchée la caméra). Dans ces cas de figure, les utilisateurs devraient pouvoir spécifier la réaction que le système doit appliquer (ignorer la demande, informer, demander confirmation, appliquer, etc.).
- La commande pourrait être amendée en fonction des contraintes posées par les programmes. Par exemple si un programme jugé prioritaire impose que le volume de la musique ne dépasse pas 50% alors toute demande de modification sera plafonnée à ce niveau.

- La commande pourrait servir à préciser le programme. Dans le cas d'un programme basé sur de l'apprentissage automatique, une commande directe contredisant l'action du système pourrait servir à mieux apprendre les réactions à appliquer. Dans le cas d'un programme réalisé par l'habitant lui-même, cela pourrait aussi servir à redéfinir le programme ou à lui apporter des corrections.

En conclusion, nous avons constaté que les problèmes d'ordonnancements sont inévitables lorsque plusieurs acteurs, humains ou programmes, peuvent agir sur l'habitat. Nous avons montré qu'il n'existe pas de façon simple d'ordonnancer les commandes de ces acteurs. Au contraire, la façon d'ordonnancer les commandes est un problème complexe dont la résolution ne peut être que contextuelle. Nous affirmons qu'elle doit dépendre des choix des habitants eux-mêmes. Par exemple, donner la priorité aux parents sur les enfants peut être un choix pour certains foyers et pas pour d'autres, ou pas à tous les âges. Ainsi les règles d'ordonnancements, tout comme les programmes, ne peuvent être définies ou personnalisés que pour un foyer particulier.

3. Approche de l'ordonnancement basée sur la programmation par l'habitant

Nous présentons dans cette section notre approche de l'ordonnancement dans l'habitat. Celle-ci s'appuie sur un système d'exploitation de l'habitat (section 3.1) et le langage CCBL (présenté dans la section 3.2). Cela nous permet de présenter dans la section 3.3 des exemples de stratégies d'ordonnancement codées avec CCBL. Enfin, nous terminons avec un point sur l'état d'implémentation du langage et les premiers retours utilisateurs (section 3.4).

3.1. Un système d'exploitation pour l'habitat

Dans l'état actuel des choses, l'habitat intelligent est constitué d'une myriade de briques de bases, appareils et services, qui peuvent fonctionner indépendamment les uns des autres. Il est possible de déployer des programmes liant ces briques les unes aux autres (ex : lorsqu'on appuie sur le bouton, alors allumer la lampe et jouer de la musique). Cependant, il n'est pas possible d'exprimer des contraintes sur l'usage de ces programmes car ceux-ci s'exécutent de façon indépendante. Il en va de même pour les interfaces utilisateurs qui permettent de commander ces briques, elles s'exécutent indépendamment les unes des autres et indépendamment des programmes. Cet état de fait rend impossible l'implémentation de stratégies d'ordonnancement telles que celles évoquées dans la section 2.3.

Nous reprenons la vision défendue dans [12, 11] qui est que l'habitat intelligent a besoin d'un système d'exploitation (OS) pour faciliter et centraliser l'utilisation des appareils et services de l'habitat. Dans cette approche, les applications ne peuvent pas parler seules et de leur propre chef aux appareils et services mais doivent en demander l'accès au système. Des stratégies d'ordonnancement peuvent alors être imaginées pour gérer ces accès. Il faut préciser ici que nous ne nous intéressons pas à

la façon dont le système est implémenté (de façon centralisée ou répartie) mais bien au rôle central qu'il occupe pour accéder aux appareils et services. On peut imaginer que, techniquement parlant, une approche décentralisée d'un tel système deviendra de plus en plus pertinente avec la multiplication des objets connectés. Actuellement, les système domotique commerciaux pour l'habitat sont techniquement basés sur des solutions centralisées [10].

Home OS [12, 11] illustre bien les trois aspects que doit couvrir un système d'exploitation pour l'habitat :

- 1) Charger de manière sécurisée des pilotes et des applications. Dans Home OS, ces derniers sont désignés par le terme de modules.
- 2) Établir des liens entre les pilotes et les applications pour transférer des données. Home OS s'appuie en particulier sur la définition de ports qui sont des abstractions de communication décrivant les rôles et les contrôles des données échangés [11].
- 3) Une gestion des priorités et des droits d'accès sur les drivers et les applications. Home OS fait le choix d'une gestion relativement simple en implémentant des règles du type : le port P peut être accédé par les utilisateurs du groupe G à l'aide du module M durant une période de temps allant de T1 à T2, avec une priorité PRI et un accès A (lecture ou écriture). Toute tentative d'accès non explicitement autorisée par une règle est refusée.

Home OS est conçu comme un système exécutant des applications produites par des professionnels. Les utilisateurs du système y sont vus comme des consommateurs d'applications. A l'inverse, nous pensons que les utilisateurs peuvent être aussi des producteurs d'applications (certes moins poussées que ce celles de professionnels). Notre perspective est celle d'un système d'exploitation permettant de combiner les applications professionnels (comme dans Home OS) et des règles produites par l'habitant (comme dans les boxes domotiques).

Les règles proposées dans Home OS se veulent simples pour être facilement appréhendables par des utilisateurs non experts en informatique. La gestion des accès dépend de seulement deux éléments : le temps et le type d'utilisateur. Cela ne représente qu'une partie du contexte, il n'est par exemple pas possible d'exprimer que les enfants ne peuvent regarder la télévision que si les parents sont présents. De plus, en cas de conflit d'accès, seul un indice de priorité établi à priori, hors contexte, permet de départager les programmes. Enfin, Home OS fait la supposition que les commandes directes des habitants au système passeront par des applications dédiées à cela. Il ne prend donc pas en compte que la commande directe peut signifier la surcharge du comportement d'un programme existant et que cela est très dépendant du contexte (au sens général) dans lequel l'utilisateur formule la commande.

Nous proposons une approche de l'ordonnancement basée sur le langage CCBL, un langage de programmation pour l'habitat que nous développons. Nous la voulons à la fois simple mais plus généraliste que celle proposée par Home OS. Nous présentons ce langage dans la section suivante. La section 3.3 discute de l'adéquation de ce

langage avec la façon de raisonner des utilisateurs et présente des premiers résultats d'évaluation. Enfin la 3.4 illustre comment CCBL peut être utilisé pour aborder le problème de l'ordonnancement dans l'habitat.

3.2. Cascading Context Based Language (CCBL)

CCBL est un langage de programmation pour utilisateurs finaux (c'est-à-dire non informaticien) conçu pour les habitats intelligents et plus généralement les environnements intelligents à petite échelle comme un appartement, une maison ou un bureau [21, 22]. Dans cet article, nous donnons une description plus détaillée du langage CCBL et introduisons la notion de programmes. CCBL a été développé pour combler les lacunes d'ECA (Event Conditions Actions) qui, bien qu'étant le paradigme de programmation dominant dans les habitats intelligents (Demeure et al., 2015), introduit des difficultés de programmation pour les non-informaticiens [15, 21]. Le domaine d'application de CCBL ainsi que son usage des contextes et relations d'Allen permet de le rapprocher du langage de description d'état proposé par [24].

La programmation par règle ECA est très pratique lorsque les automatismes sont relativement simples (par exemple « Lorsque le bouton est pressé, alors allumer les lumières de la pièce et jouer de la musique »). Cependant, [13] ainsi que [15] et [21] montrent que les habitants peuvent construire des modèles mentaux erronés du comportement d'une règle ECA notamment lorsque le déclencheur de la règle fait intervenir des verbes duratifs (par exemple la règle « Quand je suis dans la pièce, allumer la lumière » implique pour une proportion conséquente d'utilisateur que la lumière s'éteindra quand je sortirais). D'autre part, des problèmes de compréhension et de prédictibilité apparaissent lorsque le nombre de règles devient important ou que les conditions de déclanchement se complexifient [6, 19].

Le langage CCBL est quant à lui basé sur la notion de contextes (section 3.2.1) totalement ordonnés entre eux (section 3.2.2) et ayant entre eux des liens dérivés des relations d'Allen [1] (section 3.2.3). Afin de faciliter le passage à l'échelle, CCBL offre la possibilité de structurer les contextes en programmes (section 3.2.4). Ces programmes peuvent être exprimés en CCBL ou bien être des programmes externes (développés en java par exemple) pour peu qu'ils implémentent l'interface (au sens API) des programmes CCBL.

Afin d'illustrer le langage CCBL, nous proposons un cas d'étude. La Figure 3.1 représente un programme CCBL gérant le volume de la musique en fonction du contexte. Ce programme est composé de deux contextes fils : « parent à la maison » et « chaîne allumée ». Quand la chaine hifi est allumée, alors le volume de la musique est réglé par défaut à 50% (Volume : 50). Si un mode calme ou un mode rock est actif, alors le volume passe respectivement à 30% et 80%. Lorsque le bouton PLUS (respectivement MOINS) est pressé alors le volume est incrémenté (respectivement diminué) de 10%. Quand un parent est à la maison (Contexte « Parent à la maison ») alors une contrainte est posée sur le volume : il ne peut pas dépasser 50%. De plus, si un appel téléphonique est en cours alors la musique est éteinte le temps de cet appel (volume à 0). Pour des raisons de concisions, la gestion du mode n'est pas représentée.

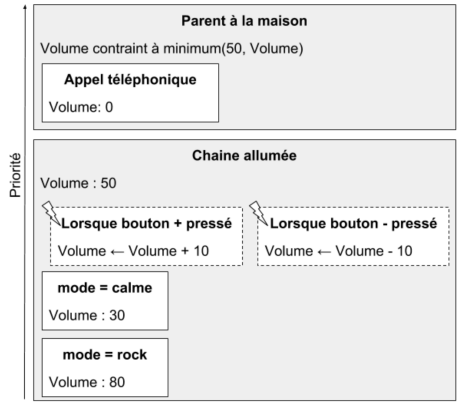

FIGURE 3.1. Programme CCBL servant de cas d'étude. La syntaxe graphique utilisée n'est qu'illustrative.

Fondamentalement, le langage CCBL permet de manipuler quatre types de données au travers des contextes pour coder des automatismes :

- Les **canaux** : un canal définie une variable qui peut être lue et écrite par CCBL (par exemple le volume dans la figure 3.1). Cette variable peut être liée à un appareil ou à un service. Si la variable est modifiée, CCBL est notifié. Si CCBL affecte une nouvelle valeur à la variable, alors l'état de l'appareil ou le service auquel elle est liée est mis à jour ;
- Les **émetteurs** : un émetteur définie une variable qui ne peut qu'être lue par CCBL. Un émetteur peut être lié à un appareil ou à un service (ex : la présence d'un parent dans la maison). Dans ce cas, si l'état de ce dernier est modifié alors la valeur de l'émetteur est mise à jour ;
- Les **expressions** : une expression définie un calcul s'appliquant sur les valeurs des émetteurs et des canaux. Lorsqu'une expression est active, sa valeur est mise à jour automatiquement lorsque la valeur d'un de ses termes (canal ou émetteur) est mis à jour (par exemple, sur la figure 3.1, « minimum (50, Volume) » est une expression dont la valeur est mise à jour lorsque la valeur de la variable Volume est mise à jour ;
- Les **événements** : un événement peut être émis par un appareil ou un service (par exemple, sur la figure 3.1, « bouton PLUS est pressé »).

Les sections suivantes montrent comment ces quatre types de données sont utilisées dans les deux types de contextes dans CCBL (voir la Figure 3.2), à savoir les **contextes d'état** et les **contextes événementiels**. Dans les deux cas les contextes appliquent des opérations a des canaux.

3.2.1. *La notion de contexte en CCBL : Les contextes d'état*

Les **contextes d'état** sont la base de CCBL. Un contexte d'état possède une condition d'activation, cette condition peut être :

- Une expression booléenne. Par exemple « mode = calme » ;
- Un évènement d'activation et un évènement de désactivation. Par exemple « un parent arrive » et « le dernier parent est parti » (ce qui dans ce cas serait équivalent à l'expression « Parent à la maison ») ;
- Un évènement d'activation et une expression booléenne. Le contexte devient actif si et seulement si l'évènement survient et que l'expression est évaluée à vraie. Il devient inactif lorsque l'expression devient fausse ;
- Un évènement d'activation, une expression booléenne et un évènement de désactivation. Le contexte devient actif si et seulement si l'évènement d'activation survient et que l'expression est évaluée à vraie. Il devient inactif lorsque l'expression devient fausse ou bien lorsque l'évènement de désactivation survient.

Les contextes d'état permettent de définir des opérations d'affectation et de contraintes sur les canaux. Une opération d'affectation affecte la valeur d'une expression à un canal (par exemple « Volume : 50 »). Lorsque la valeur de l'expression change alors la valeur du canal change en conséquence. Comme ce même canal peut être utilisé dans d'autres expressions cela peut provoquer une cascade de mises à jour. Afin d'éviter les problèmes de boucles infinies, l'interpréteur CCBL n'autorise pas la définition d'une opération d'affectation si elle introduit un cycle (par exemple « A dépend de B qui dépend de C qui dépend de A »).

Une opération de contrainte sur un canal signifie qu'on associe au canal une expression qui sera évaluée après l'opération d'affectation associée à ce même canal. Dans le cas d'étude présenté dans la Figure 3.1, il s'agit par exemple de l'opération « Volume contraint à minimum (50, Volume) ».

3.2.2. *La notion de contexte en CCBL : Les contextes événementiels*

Les **contextes événementiels** sont constitués d'un évènement (ex : « le bouton PLUS est pressé ») et d'opérations de surcharges sur les canaux (ex : « Volume ← Volume + 10). Lorsque l'évènement survient alors les opérations de surcharge sont appliquées. Une opération de surcharge s'applique à un canal, elle surcharge l'expression associée à l'opération d'affectation liée à ce canal par la valeur de l'expression de surcharge au moment du déclenchement de l'évènement. Par exemple, dans la Figure 3.1, si le mode est « calme » et qu'on appuie sur le bouton plus, alors le volume sera incrémenté de 10.

La section suivante discute de la priorité entre les contextes et de l'implication sur les opérations.

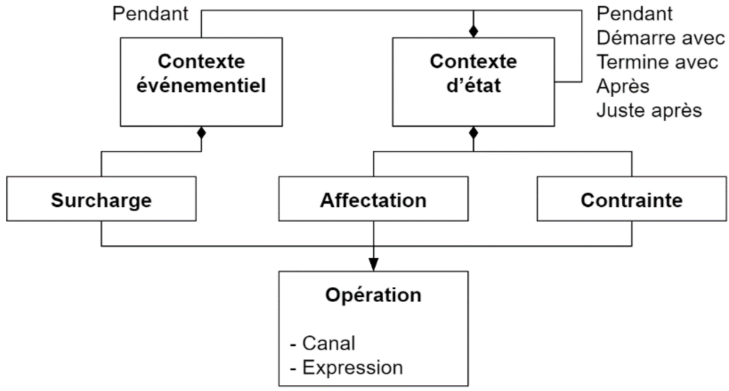

FIGURE 3.2. Contextes et opérations dans CCBL. Un contexte d'état peut être lié à d'autres contextes d'état par des relation d'Allen. Il peut être lié à des contextes événementiels mais seulement par la relation « pendant que ». Un contexte d'état peut définir des opérations d'affectation et de contraintes. Un contexte événementiel ne peut définir que des opérations de surcharges.

3.2.3. *Ordre total sur les contextes et prédictibilité des opérations*

Dans le langage CCBL, les contextes sont totalement ordonnés. Cet ordre nous permet d'ordonner les opérations portant sur un même canal. Cette façon de procéder a été inspirée par le langage de style CSS (Cascading Style Sheet) définit par Lie [16], elle se rapproche également de la notion de subsumption [3]. La Figure 3.3 illustre cela. Pour chaque canal λ, il y a un ensemble d'opérations qui s'applique. Elles sont définies dans des contextes d'état ou des contextes événementiels. A un moment donné, seule une partie de ces contextes sont actifs. Ce sont les opérations portant sur le canal λ, contenues dans ces contextes actifs, qui sont représentées par la pile de la Figure 3.3.

Les opérations sont ordonnées suivant la priorité associée au contexte qui les définit. Sur l'exemple de la figure 3.3, on se retrouve alors avec l'opération d'affectation A définie comme l'opération d'affectation ayant le plus haut niveau de priorité dans la pile. Toutes les opérations moins prioritaires que A sont ignorées. La valeur affectée au canal λ est alors calculée en appliquant les contraintes C à 1 sur la valeur de l'expression définie dans l'opération d'affectation A. Si la valeur d'une des expressions associées à A ou aux contraintes 1 à C est modifiée alors le calcul de la valeur de λ est mise à jour. C'est ce mécanisme d'affectation des valeurs aux canaux en fonction des priorités que nous appelons Cascade dans CCBL.

Enfin, les opérations de surcharges 1 à S peuvent être appliquées si le contexte événementiel correspondant est activé. Dans ce cas, la valeur de l'expression de l'opération de surcharge i est calculée et son résultat surcharge l'expression de A. Il est à noter que la surcharge de A continue de s'appliquer tant que le contexte définissant A est actif et tant qu'une nouvelle surcharge n'est pas appliquée à A. Lorsque le contexte dans lequel A est défini devient inactif, la surcharge est « oubliée ».

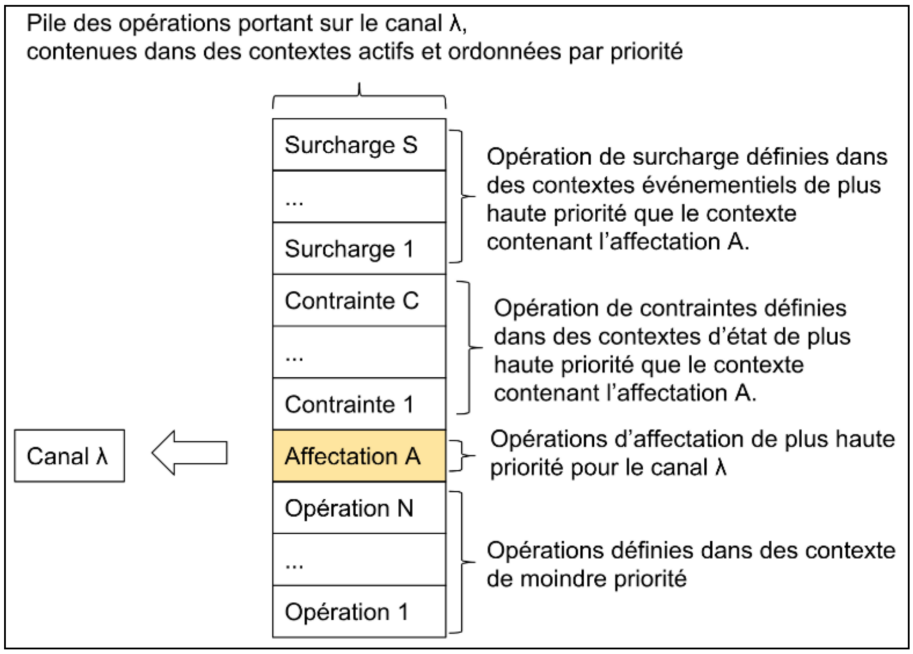

FIGURE 3.3. Ordonnancement des opérations sur un même canal selon la priorité associée aux contextes de ces opérations. La valeur affectée au canal est celle de l'opération d'affectation (A) de plus haute priorité, en appliquant les contraintes définies dans les contextes de priorité supérieur à cette opération d'affectation. Les opérations de surcharges 1 à S peuvent surcharger la valeur d'affectation définie dans A.

La section suivante présente les différentes relations qui peuvent exister entre les contextes. Elle explicite aussi comment est calculé l'ordre de priorité des contextes.

3.2.4. *Les relations d'Allen entre les contextes*

CCBL a été conçu pour éviter un des écueils de la programmation ECA, à savoir l'accumulation non structurée de règles. CCBL fournit plusieurs façons d'organiser les contextes entre eux en s'appuyant sur des propriétés temporelles dérivées de l'Algèbre d'Allen [1]. Nous présentons ici la sémantique de ces différentes relations et illustrons comment chacune d'elle peut être utilisée. Nous terminons en explicitant comment ses relations sont utilisés pour le calcul de priorité entre les opérations définies dans les contextes.

SC pendant C. Cette relation permet d'exprimer un sous cas ou une exception. Le contexte SC n'est considéré que lorsque le contexte C est actif. La sémantique est la suivante : tant que le contexte C est actif, les opérations spécifiées par C sont appliquées. Lorsque SC est aussi actif alors les opérations de SC sont appliquées de

façon prioritaire sur celles de C. Par exemple, si C affecte la couleur blanche à la lampe L et que SC affecte la couleur rouge, alors c'est la couleur rouge qui sera choisie. Si SC devient inactif alors ses opérations ne sont plus appliquées, seules restent les opérations définies par C. Dans notre exemple, la lampe redevient alors blanche.

Un scénario illustrant l'usage de cette relation peut être le suivant : *quand Martin est chez lui (contexte C), il veut que de la musique soit jouée à volume normal, sauf s'il parle au téléphone (contexte SC). Dans ce cas, Martin veut que la musique soit jouée à volume faible.*

En utilisant la relation SC pendant C, l'habitant programmeur exprime seulement un cas général (contexte C => musique jouée à volume normal) et un cas particulier à ce cas général (contexte SC => musique jouée à volume faible). CCBL basculera automatiquement entre le cas général et le cas particulier en fonction du contexte. En particulier, contrairement à ECA, l'utilisateur n'a pas besoin de spécifier que la musique revient à volume normal lorsque l'appel téléphonique prend fin. Or, il est fréquent que les programmeurs oublient ce cas lorsqu'ils programment avec ECA [13, 15, 21].

SC démarre avec C. Cette relation est une spécialisation de la relation SC pendant C. Le contexte SC n'est considéré qu'au début du contexte C. C'est-à-dire que CCBL vérifie si SC est actif seulement au moment où C devient actif. Après cela le contexte SC n'est plus considéré tant que C est actif.

Un scénario illustrant l'usage de cette relation peut être le suivant : *quand Alice est chez elle (contexte C), le système joue la musique de sa radio préférée. Cependant, lorsqu'elle rentre chez elle et qu'elle a reçu plusieurs notifications, alors le système commence par lui énumérer les notifications reçues jusqu'à ce qu'il n'y en ait plus ou qu'elle dise stop (contexte SC). La musique est alors jouée.*

SC termine avec C. Cette relation est aussi une spécialisation de la relation C pendant SC. Le contexte SC ne peut, dans ce cas, qu'être défini par un événement déclencheur et ne comporte pas de fin explicite (il se termine avec le contexte C).

Un scénario illustrant l'usage de cette relation peut être le suivant : *quand Alice est chez elle les matins des jours de travail (contexte C), elle écoute les informations à la radio. Si le système détecte qu'elle risque d'arriver en retard au travail (démarrage du contexte SC) alors le système joue une musique spéciale (la danse du sabre) jusqu'à ce qu'elle quitte la maison (fin du contexte C).*

C2 juste après C1. Cette relation exprime que le contexte C2 n'est considéré par CCBL que s'il fait immédiatement suite au contexte C1. C'est-à-dire que ça n'est que juste après que C1 soit devenu inactif que CCBL évaluera si C2 doit être activé ou pas. Si C2 ne peut pas être activé à ce moment-là alors il ne le sera pas du tout (tant que C1 ne redevient pas à nouveau actif).

Nous illustrons cette relation avec un programme du type (C2 juste après C1) pendant C : *quand Alice regarde la télévision (contexte C), si elle est au téléphone (contexte C1) alors la lecture de la télévision est mise en pause. Comme il lui arrive*

d'arpenter l'appartement pendant qu'elle téléphone, elle veut que la télévision reste en pause jusqu'à ce qu'elle soit de retour sur son canapé (contexte C2).

C2 après C1. Cette relation exprime que C2 n'est considéré qu'après que C1 soit devenu inactif, mais pas nécessairement juste après. Nous illustrons cette relation avec l'exemple suivant : *Alice fait parfois des soirées avec des amis chez elle (contexte C1). Après la soirée, elle veut être sûr que ses amis sont bien rentrés (elle craint les accidents de voiture). Pour cela elle utilise une lampe particulière qui est allumée en blanc pendant la soirée (contexte C1). Cette lampe devient orange après la soirée (contexte C1) puis elle devient verte quand tous ses amis sont bien arrivés chez eux (contexte C2).*

CCBL n'implémente pas les relations de chevauchement (C1 chevauche C2) ni d'égalité (C1 en même temps que C2). En effet, il n'est pas possible d'exprimer à priori que C1 chevauchera C2, cela ne peut être observé qu'à postériori. Il en va de même pour la relation d'égalité.

Le calcul de la priorité des contextes s'effectue selon les règles suivantes :

- Si SC pendant que C, alors SC est plus prioritaire que C ;
- Si SA après SC alors SA plus prioritaire que SC ;
- Si SJA juste après SA alors SJA plus prioritaire que SA ;
- Si SS démarre avec C alors SS plus prioritaire que SJA ;
- Si SE termine avec C alors SE plus prioritaire que SS ;
- Enfin, pour les cas où les relations ne permettent pas d'ordonner les contextes (ex : SC1 et SC2 pendant C) alors l'ordre de déclaration est pris en compte (SC2 > SC1).

La section suivante montre comment les contextes CCBL peuvent être groupés en programmes afin de faciliter l'expression de comportements et de permettre l'utilisation de programmes autres que CCBL.

3.2.5. *La notion de programme en CCBL*

CCBL structure les contextes en programmes. Un programme est constitué d'un contexte d'état racine, d'un ensemble d'imports (canaux, émetteurs et événements) et d'un ensemble d'exports (canaux, émetteurs et événements). CCBL impose la définition d'un seul programme racine.

Chaque programme peut définir et instancier des sous-programmes. L'instanciation nécessite de spécifier les imports. Le sous-programme alors instancié peut être vu comme une boite noire et prend la forme d'un contexte d'état définissant des opérations d'affectation sur les canaux qu'il manipule (cf. Figure 3.4). Les exports du sous-programme instancié deviennent autant de nouveaux canaux, émetteurs et événements à l'intérieur du programme qui l'instancie. Enfin, un canal de type booléen permettant de démarrer ou d'arrêter l'instance est ajouté au programme qui instancie.

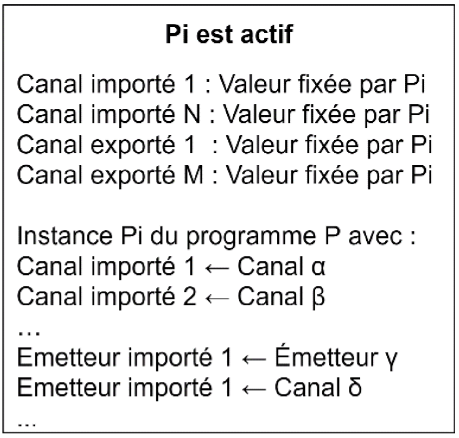

FIGURE 3.4. Exemple d'instance Pi d'un programme P. L'instance est intégrée comme un contexte d'état. Les canaux importés et exportés sont affectés par Pi. Notons qu'un canal peut être affecté comme émetteur importé, il ne sera alors accessible qu'en lecture seule au sein du sous-programme.

Ces programmes peuvent être écrits en CCBL (à l'aide de contextes, de canaux, etc.) ou dans un autre langage (en fournissant une API compatible). Ils peuvent, en conséquence, proposer une implémentation quelconque, par exemple basée sur de l'apprentissage automatique. Une autre possibilité est que le programme soit une télécommande (graphique, vocale ou autre) permettant à l'utilisateur de contrôler des appareils et des services via des canaux, émetteurs et événements CCBL. De cette façon, l'interaction par commande directe se trouve intégrée au système de façon similaire aux autres programmes.

La Figure 3.5 reprend l'exemple du contrôle du volume sonore de la chaîne hifi décrit dans la Figure 3.1 en y ajoutant un programme lié à une télécommande logicielle fonctionnant sur un smartphone. Dans cet exemple, lorsque le télécommande est active, elle « masque » les opérations du contexte « Chaîne allumée » mais reste moins prioritaire que les opérations du contexte « Parent à la maison ». Ainsi, lorsque la télécommande est lancée sur le téléphone, elle prend la main mais le volume reste contraint à un maximum de 50% si un des parents est à la maison et sera fixé à 0 si un appel téléphonique est en cours. Notons qu'en déplaçant le contexte lié à la télécommande (« Télécommande volume est actif »), il devient facile de changer ce comportement. Par exemple, en déplaçant ce contexte à l'intérieur du contexte « Chaîne allumée », entre le contexte « mode = calme » et les contextes événementiels, on peut alors continuer d'utiliser les boutons PLUS et MOINS pour manipuler le volume (en plus de la télécommande elle-même).

La section suivante montre comment le problème de l'ordonnancement peut être aborder à l'aide des contextes et des programmes en CCBL.

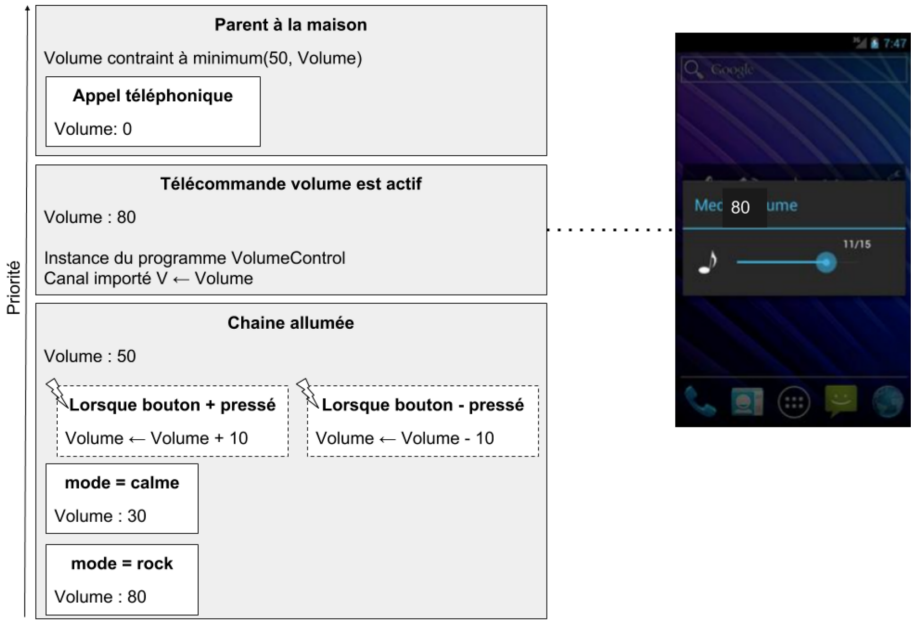

FIGURE 3.5. Exemple de gestion du volume de la musique avec un programme lié à une télécommande sur téléphone. La télécommande est ici prioritaire sur le contexte « chaîne allumée » mais elle l'est moins que le contexte « Parent à la maison ».

3.3. GESTION DE L'ORDONNANCEMENT AVEC CCBL

Le problème de l'ordonnancement dans l'habitat est celui de l'expression de buts divergents par différents acteurs (humains ou programmes). Il s'agit pour le système de définir comment ces buts doivent être interprétés, si certains sont plus prioritaires que d'autres et en fonction de quoi. Nous avons montré dans la section 2.3 qu'il n'existe pas de façon simple d'ordonnancer les choses. Au contraire, les stratégies d'ordonnancement sont contextuelles et doivent dépendre du choix des habitants.

Nous montrons dans cette section comment CCBL peut être utilisé pour définir différentes stratégies d'ordonnancement. Nous nous appuierons pour cela sur les propriétés du langage que sont l'ordre total sur les contextes, le mécanisme de cascade permettant l'affectation de la valeur des canaux et le mécanisme de surcharge des valeurs d'affectations.

Il est possible, et relativement aisé, de spécifier des stratégies d'ordonnancement différentes selon les contextes. Prenons l'exemple de la sécurité dans la maison tel qu'illustré dans la Figure 3.6. Nous supposons que l'habitat est constitué d'au moins une porte qu'on peut verrouiller et de lumières. Par défaut (contexte racine), la porte est verrouillée et les lumières éteintes. Cela peut être modifié par les règles codées dans le comportement standard et par le contexte de sécurité. Pour assurer le fait que la sécurité

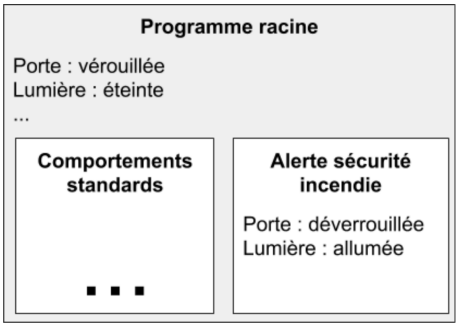

FIGURE 3.6. Exemple de gestion prioritaire de la sécurité. Quels que soient les comportement standards, CCBL assure que si le contexte « Alerte sécurité incendie » est actif alors la porte est déverrouillée et les lumières allumées.

est prioritaire, il suffit d'encapsuler les comportements standards liés à l'habitat dans un contexte ad-hoc et d'ajouter le contexte « Alerte sécurité incendie ». Tous les deux se déroulent « pendant que » le programme racine est actif, mais le contexte de sécurité est plus prioritaire car déclaré après le contexte « Comportement standard » (ce qui est symbolisé par le fait qu'il se trouve à sa droite sur la figure 3.6). Ainsi, quels que soient les automatismes et commandes directes des comportements standards, CCBL assure que si le contexte « Alerte sécurité incendie » est actif alors la porte est déverrouillée et les lumières allumées.

De façon similaire, il est possible d'ordonner les programmes selon leur « propriétaire » ou auteur (parents, enfants, invités, programme d'apprentissage automatique, etc.). Dans le cas où un programme basé sur l'apprentissage automatique est à l'œuvre, cela peut permettre aux habitants d'exprimer des « gardes fous » afin d'éviter des comportements indésirables de l'habitat comme cela est décrit par Yang et Newman (2013). Notons cependant que si notre approche permet aux habitants de garder le contrôle, elle ne permet pas, en elle-même, d'améliorer le processus d'apprentissage automatique.

La Figure 3.7 décrit un exemple où la sécurité prime toujours sur tout le reste mais où la priorité entre les programmes des parents et des enfants peut changer. En mode « normal », ce sont les parents qui ont la priorité sur le contrôle des appareils et des services, mais en mode « boom », lors des fêtes organisées par les enfants, alors ce sont les programmes des enfants qui deviennent prioritaires. L'expression de ce changement de priorité en fonction du contexte est exprimée par le simple changement d'ordre de déclaration. Dans le contexte « Mode = normal », les programmes des enfants des enfants sont définis dans un contexte « Enfants » qui se trouvent à gauche du contexte « Parents ». Cela a pour conséquence, d'après les règles de priorité, de rendre les programmes des parents prioritaires sur ceux des enfants. Inversement, dans le contexte « Mode = boom », ce sont les programmes des enfants qui deviennent prioritaires sur ceux des parents.

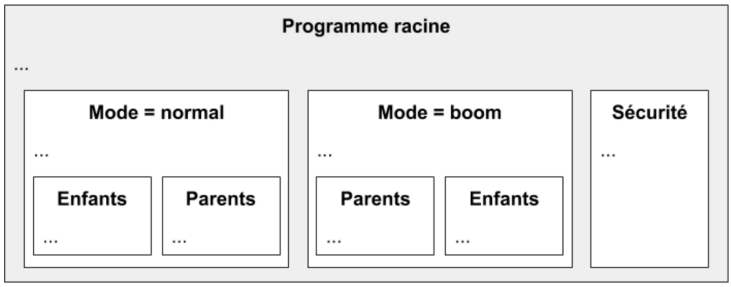

FIGURE 3.7. Exemple d'ordonnancement dépendant du contexte. En mode normal les programmes des parents sont prioritaires, mais c'est l'inverse lors des fêtes organisés par les enfants. Seule la sécurité reste prioritaire dans tous les cas.

Notons que les deux programmes « Parents » sont des instances du même programme, il n'est donc pas nécessaire de spécifier deux fois les choses (et il en est de même pour les deux programmes « Enfants »).

Les exemples précédents illustrent comment programmer des stratégies d'ordonnancement à l'aide de CCBL. D'une façon générale, il faut noter que programmer des stratégies d'ordonnancements est similaire à programmer des automatismes avec CCBL, nous pensons que c'est une des forces de notre approche. L'exemple donné dans la Figure 3.5 montre d'ailleurs un problème qui peut être vu comme l'ordonnancement entre un programme de commande directe (la télécommande sur smartphone) et des automatismes dans la maison.

3.4. IMPLÉMENTATION ET PREMIERS RETOURS SUR CCBL

Nous avons implémenté un interpréteur CCBL dans le langage TypeScript [(5)] et avons publié la bibliothèque correspondante sur GitHub [(6)]. Les exemples d'ordonnancement présentés dans cet article ont été implémenté et testés sur des environnements simulés. L'objectif n'est cependant pas de populariser CCBL tel quel. Nous travaillons sur un éditeur graphique du langage qui est encore en phase de développement.

Nous avons voulu tester auprès d'utilisateur (programmeurs et non programmeurs) si la logique sous-jacente à CCBL était bien comprise, en particulier les notions de contexte et de cascade. Nous avons pour cela réaliser une expérimentation dont les résultats sont présentés dans [21]. L'objectif de cette étude était de vérifier si les utilisateurs pouvaient utiliser aussi facilement CCBL qu'un langage de type ECA (qui est la forme de langage dominante pour programmer l'habitat). Nous avons demandé aux participants de programmer quatre automatismes de complexité croissante. Notre hypothèse était que CCBL serait aussi performant qu'ECA pour des problèmes simples mais plus performant pour des problèmes complexes.

[(5)] `https://www.typescriptlang.org/`
[(6)] `https://github.com/AlexDmr/ccbl-js`

Nous avons confronté 11 programmeurs et 10 non-programmeurs à 4 exercices de complexité croissante. Ces exercices étaient basés sur les habitats de Martin et Alice. L'habitat de Martin était constitué de détecteurs de présence (indiquant la présence d'Alice et/ou Martin et indiquant si l'un des deux est sur le canapé), d'un lecteur de musique dont le volume était programmable et enfin d'une lampe qui pouvait être allumée de différentes couleurs. L'habitat d'Alice était quant à lui équipé d'un seul capteur de présence indiquant si Alice était chez elle ou pas. Enfin, Alice pouvait renseigner un statut indiquant si elle était disponible ou non, indépendamment de sa présence chez elle (à l'image d'un statut de réseau social modifiable sur un smartphone). Les quatre exercices à programmer étaient les suivants :

- **Exercice 1** : Martin veut que la lampe soit allumée si et seulement s'il est chez lui (s'il n'est pas chez lui la lampe doit être éteinte).
- **Exercice 2** : Martin veut que le volume de la musique soit faible quand il est chez lui, sauf s'il est sur le canapé, auquel cas le volume de la musique doit être normal. Quand il n'est pas à la maison, Martin veut que la musique soit éteinte.
- **Exercice 3** : Martin veut que le volume de la musique soit normal quand il est chez lui, sauf s'il parle au téléphone. Dans ce cas, le volume de la musique doit être faible. Enfin, la musique doit être éteinte si Martin n'est pas chez lui.
- **Exercice 4** : Martin veut utiliser sa lampe pour savoir quand Alice est chez elle. Quand Alice est chez elle et qu'elle est disponible, Martin veut que la lampe soit allumée en vert. Quand Alice est chez elle et qu'elle n'est pas disponible, il veut que la lampe soit allumée en orange. Quand Martin reçoit Alice chez lui il veut que la lampe soit allumée en blanc. Enfin, quand Martin n'est pas chez lui il veut que la lampe reste éteinte.

Les résultats de ces exercices sont synthétisés sur la Figure 3.8. Nous avons bien observé un accroissement du nombre d'erreurs faites avec ECA à mesure que la complexité des exercices augmentait. Seuls un programmeur et un non-programmeur ont correctement programmé l'exercice 4 avec le langage ECA. A contrario, seuls deux non-programmeurs n'ont pas correctement programmé ce même exercice avec CCBL.

L'analyse des programmes produits ainsi que les discussions avec les participants ont révélés que de nombreux participants avaient eu tendance à raisonner en termes de « cas général » et « cas particulier ». De plus, s'ils spécifiaient généralement bien l'état que le système devait adopter lorsqu'un nouveau contexte était détecté (par exemple lorsque Martin entre chez lui), ils ne faisaient pas de même pour gérer la réaction du système lorsqu'un contexte n'était plus vrai (par exemple lorsque Martin raccroche son téléphone). Or, ce sont des façons de raisonner qui sont mieux prises en compte par CCBL que par ECA.

Il est apparu que les utilisateurs comprenaient bien la notion de contextes d'état et d'imbrication de contextes (relation d'Allen « pendant »). La notion de cascade semblait moins bien comprise et certains utilisateurs ont formuler leurs solutions de manière à ne pas en dépendre (mais leurs solutions étaient correctes).

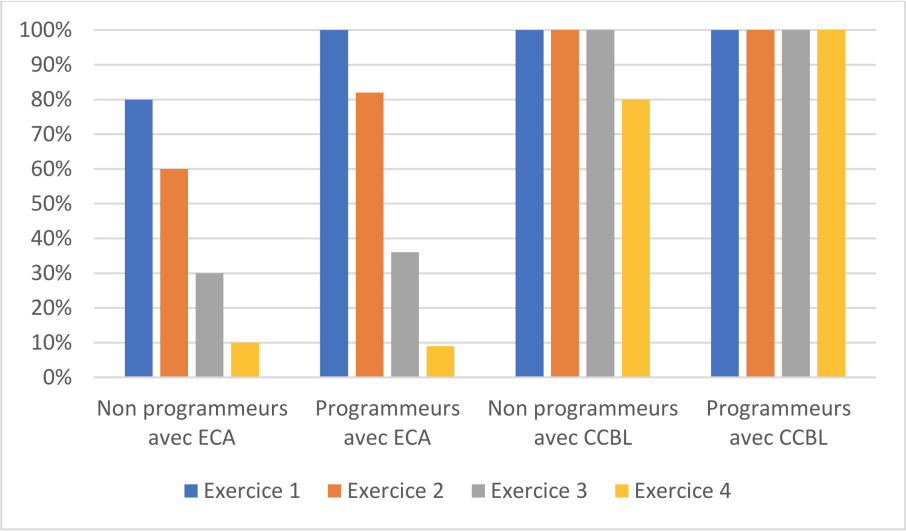

FIGURE 3.8. Taux de réussite des 10 programmeurs et 11 non-programmeurs face à 4 exercices de complexités croissante en utilisant les langages ECA et CCBL.

Ces résultats sont encourageants mais doivent encore être confirmés pour des problèmes plus complexes et dans le cas de l'expression de stratégies d'ordonnancement par des utilisateurs.

4. Conclusion, limites et perspectives

Nous abordons dans cet article l'une des difficultés des habitats intelligents : l'accès (et le contrôle) des appareils et services présents dans l'habitat. La coexistence de plusieurs acteurs (habitants et programmes) qui agissent sur les mêmes ressources impose d'ordonnancer ces accès afin d'éviter l'apparition de situations indésirables. Les différents acteurs et leur façon d'agir sur l'habitat sont tous pertinents dans certains contextes, mais ils peuvent avoir des buts différents, il est donc nécessaire de pouvoir les ordonnancer.

Nous avons montré que l'ordonnancement est un problème complexe, qui ne peut être résolu a priori et qui dépend du contexte. Nous défendons l'idée que le problème doit être abordé sous l'angle d'un système d'exploitation pour la maison comme cela est proposé par Home OS [12, 11]. Nous proposons cependant une approche différente, basée sur les travaux dans le domaine du développement par l'utilisateur final (voir [5]) et inspirée de la pratique des boxes domotiques.

Pour spécifier l'ordonnancement au sein de l'habitat, nous proposons donc d'utiliser un langage développé pour permettre aux habitants d'exprimer des programmes en fonction de contextes, à savoir CCBL [21, 22]. Ce langage permet à la fois de programmer des automatismes pour l'habitat et d'intégrer des programmes extérieurs

(que ce soit des programmes de type télécommande ou de type apprentissage automatique). Nous avons illustré comment CCBL peut être utilisé pour programmer différentes stratégies d'ordonnancement et comment cela est similaire à programmer des automatismes avec CCBL. Nous pensons que cela est une force de notre approche, les connaissances acquises pour programmer des automatismes étant réutilisables pour la programmation de règles d'ordonnancement et *vice versa*.

Il existe cependant des limites à notre approche. En premier lieu, la multiplication des capteurs et la complexification des données qu'ils peuvent produire peut rendre impraticable l'expression de certains contextes pertinents par les utilisateurs (par exemple un contexte qui impliquerait la combinaison des données issues d'une dizaine de capteurs dont certains produiraient des données « complexes »). Pour faire face à cette limite, une approche envisageable serait de s'appuyer sur des algorithmes d'apprentissage automatique. Les algorithmes auraient la charge de proposer des contextes de haut niveau aux utilisateurs. Ces derniers pourraient alors les utiliser dans leurs programmes CCBL. Vice versa, l'analyse des programmes CCBL produits par les utilisateurs pourraient guider les algorithmes d'apprentissages en les orientant vers ce qui semble intéresser les utilisateurs.

En second lieu, même en disposant de contextes de haut niveau, tous les ordonnancements possibles ne sont pas aisément exprimables par un utilisateur à l'aide de CCBL. Supposons par exemple qu'un foyer équipé en domotique reçoive des invités possédants eux-mêmes un tel système et qu'il souhaite intégrer leurs préférences en termes musicales et d'éclairages. Cela serait possible à l'aide de CCBL mais pourrait se révéler fastidieux (il faudrait faire correspondre les lumières et les instruments de lectures musicaux entres les deux habitats puis exprimer l'ordonnancement souhaité). Plus généralement, nous avons montré dans cet article que CCBL était bien adapté pour exprimer certaines formes d'ordonnancement mais il serait intéressant de pouvoir caractériser plus formellement les différentes stratégies d'ordonnancement afin de discuter de la possibilité (et de la facilité) de leur mise en œuvre à l'aide de CCBL.

Outre les limites précédemment évoquées, les perspectives de ce travail sont multiples. Permettre aux habitants de spécifier l'ordonnancement au sein de leur foyer nécessite de mettre à disposition un langage mais également les outils pour l'utiliser. A court terme, nous voulons donc proposer un outil interactif pour le langage CCBL. Cet outil disposera à la fois d'un éditeur basé sur une syntaxe graphique mais également des environnements virtuels sur lesquels les programmes pourront être exécutés. Une fois cet outil développé nous mènerons des tests utilisateurs pour évaluer l'utilisabilité de notre solution. Il serait aussi intéressant de réfléchir à une intégration avec des langages de descriptions de contexte tel que celui présenté par [24] afin d'enrichir l'expressivité de CCBL.

A plus long terme, nous souhaitons travailler sur la façon de présenter les programmes CCBL à l'utilisateur afin qu'il comprenne mieux ce qui se passe dans son environnement (pourquoi telle chose se produit et pourquoi telle autre ne se produit pas). La difficulté est de présenter cela dans l'environnement de l'utilisateur sans alourdir sa charge mentale de manière significative. D'autre part, nous souhaitons étudier

comment les utilisateurs pourrait lever des « exceptions », par exemple pour communiquer au système que son comportement est inadéquat dans le contexte présent. Le traitement de ces exceptions sur le moment et/ou a postériori pourrait aider à faire évoluer les programmes vers une meilleure adéquation aux besoins des habitants.

Bibliographie

[1] J. F. Allen, « Maintaining knowledge about temporal intervals », *Communications of the ACM* **26** (1983), nº 11, p. 832-843.

[2] C. Björkskog, « Human computer interaction in smart homes », *Helsinki, Finland* (2009), p. 1.

[3] R. A. Brooks, « Intelligence without representation », *Artificial intelligence* **47** (1991), nº 1-3, p. 139-159.

[4] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu & C. Dixon, « Home automation in the wild : challenges and opportunities », in *proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, p. 2115-2124.

[5] M. M. Burnett & C. Scaffidi, « End-user development », *Encyclopedia of human-computer interaction* (2011).

[6] J. Cano, G. Delaval & E. Rutten, « Coordination of ECA rules by verification and control », in *International Conference on Coordination Languages and Models*, Springer, 2014, p. 33-48.

[7] J. Coutaz & J. L. Crowley, « A first-person experience with end-user development for smart homes », *IEEE Pervasive Computing* **15** (2016), nº 2, p. 26-39.

[8] J. Coutaz, A. Demeure, S. Caffiau & J. L. Crowley, « Early lessons from the development of SPOK, an end-user development environment for smart homes », in *Proceedings of the 2014 acm international joint conference on pervasive and ubiquitous computing : Adjunct publication*, 2014, p. 895-902.

[9] S. Davidoff, M. K. Lee, C. Yiu, J. Zimmerman & A. K. Dey, « Principles of smart home control », in *International conference on ubiquitous computing*, Springer, 2006, p. 19-34.

[10] A. Demeure, S. Caffiau, E. Elias & C. Roux, « Building and using home automation systems : a field study », in *International Symposium on End User Development*, Springer, 2015, p. 125-140.

[11] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu & P. Bahl, « An operating system for the home », in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, p. 337-352.

[12] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu & V. Bahl, « The home needs an operating system (and an app store) », in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, p. 1-6.

[13] E. Fontaine, A. Demeure, J. Coutaz & N. Mandran, « Retour d'expérience sur KISS, un outil de développement d'habitat intelligent par l'utilisateur final », in *Proceedings of the 2012 Conference on Ergonomie et Interaction homme-machine*, 2012, p. 153-160.

[14] V. Guivarch, J. F. De Paz, G. Villarrubia, J. Bajo, A. Péninou & V. Camps, « Hybrid system to analyze user's behaviour », in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2016, p. 1-5.

[15] J. Huang & M. Cakmak, « Supporting mental model accuracy in trigger-action programming », in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, p. 215-225.

[16] H. W. Lie & B. Bos, *Cascading style sheets : Designing for the web, Portable Documents*, Addison-Wesley Professional, 2005.

[17] S. Mennicken & E. M. Huang, « Hacking the natural habitat : an in-the-wild study of smart homes, their development, and the people who live in them », in *International conference on pervasive computing*, Springer, 2012, p. 143-160.

[18] S. Mennicken, J. Vermeulen & E. M. Huang, « 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*, 2014, p. 105-115.

[19] C. Nandi & M. D. Ernst, « Automatic trigger generation for rule-based smart homes », in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016, p. 97-102.

[20] A. kishore Ramakrishnan, D. Preuveneers & Y. Berbers, « Enabling self-learning in dynamic and open IoT environments », *Procedia Computer Science* **32** (2014), p. 207-214.

[21] L. Terrier, A. Demeure & S. Caffiau, « Ccbl : A language for better supporting context centered programming in the smart home », *Proceedings of the ACM on Human-Computer Interaction* **1** (2017), n° EICS, p. 1-18.

[22] L. Terrier, A. Demeure & S. Caffiau, « CCBL : A new language for End User Development in the Smart Homes », *Proceedings of IS-EUD* (2017), p. 82-87.

[23] A. Vianello, Y. Florack, A. Bellucci & G. Jacucci, « T4Tags 2.0 : A Tangible System for Supporting Users' Needs in the Domestic Environment », in *Proceedings of the TEI'16 : Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, 2016, p. 38-43.

[24] N. Volanschi, B. Serpette, A. Carteron & C. Consel, « A language for online state processing of binary sensors, applied to ambient assisted living », *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* **2** (2018), n° 4, p. 1-26.

[25] R. Yang & M. W. Newman, « Learning from a learning thermostat : lessons for intelligent systems for the home », in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, 2013, p. 93-102.

Abstract. — The text is about the problem of the scheduling of the actions applied to the actuators of a smart home. These actions can be triggered either by inhabitants or by programs encoding automatisms. We show that this is a complex problem that cannot be solved a priori. On the contrary, it depends on the context. We defend the idea that this problem should be tackle from the angle of an operating system which scheduling engine would be based on CCBL (Cascading Contexts Based Language). CCBL is an end-user programming language for the smart home that enable inhabitants to programs automatisms based on devices and services. We provide several examples of scheduling strategies programmed with CCBL. We show using CCBL to program such strategies is not fundamentally different than programming mere automatisms. Hence, the skills acquired in one of the tasks will be reusable in the other.

Keywords. — Smart Home, DSL, CCBL, End User Programming, End User Development, scheduling.
