Document :	*Modélisation du contexte et Adaptation* dans CONTINUUM
Sous-tâches :	2.1 et 2.2
Numéro des Livrables :	D2.1 et D2.2
Date	20/09/2010
Rédacteurs :	*Gaëtan Rey(I3S), Jean-Yves Tigli(I3S), Stéphane Lavirotte(I3S), Nicolas Ferry(I3S), Sana Fathallah (I3S), Joëlle Coutaz(LIG), Emeric Fontaine(LIG), Fabrice Jouanot (LIG), Anis Benyelloul (LIG), Marie-Christine Rousset(LIG), Philippe Renevier(I3S), Anne-Marie Pinna-Dery (I3S),Vincent Hourdin (MobileGov),*
Coordinateurs :	Gaëtan Rey

Sommaire

1 Introduction

1.1 Système ambiant et ses contraintes

1.1.1 Définition d'un système ambiant :

En 1991, Mark Weiser [Weiser93] introduit une première vision de l'informatique ambiante (« ubiquitous computing »). Il parlait alors d'une informatique invisible présente en tout lieu, en toute chose. Aujourd'hui, avec la miniaturisation des matériels informatiques, de nombreux objets informatisés se trouvent dissous dans notre quotidien et font peu à peu disparaître l'idée de l'ordinateur personnel comme le seul objet informatisé, ou encore de l'ordinateur comme l'assistant numérique universel. Ainsi, un système d'informatique ambiante est « multi » :

- « multi-utilisateurs » puisque dans un même espace de nombreux utilisateurs peuvent être amenés à utiliser les mêmes ressources et à communiquer de manière concurrente,
- « Multi-dispositifs » car de nombreux objets physiques potentiellement informatisés et communicants peuvent nous entourer,
- « Multi-environnements », puisqu'ils sont mobiles tout comme les utilisateurs et peuvent changer d'emplacement et donc d'environnement,
- « Multi-applications », puisque de nombreuses applications reposant sur ces objets communicants peuvent être conçues.

L'informatique ambiante repose donc sur un ensemble d'entités en interaction et sont incluses dans l'environnement (cf **Figure 1**). On entend par environnement l'ensemble des objets, des personnes et des systèmes informatiques du monde physique.

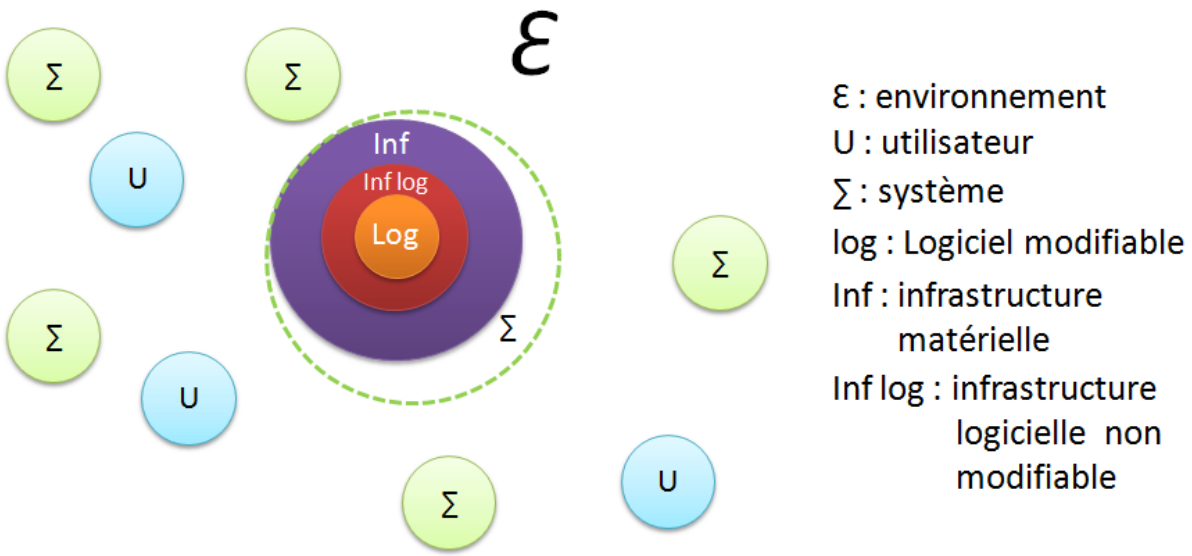


Figure 1. Modèle générale de l'informatique ambiante

D'un point de vue informatique, ces entités sont de deux catégories [Rey05]:

- Une entité ambiante peut être un « être vivant », c'est-à-dire un utilisateur ou plus généralement une entité dotée de capacités sensori-motrices, voire cognitives.

- Elle peut-être aussi un système informatisé. Ces systèmes reposent alors sur une infrastructure d'objets physiques appelés aussi dispositifs, qui peuvent offrir à leur tour une interface logicielle, appelée infrastructure logicielle, pour le reste du système informatisé. C'est sous ces contraintes que la partie logicielle modifiable du système permet *in fine* de mettre en œuvre de nouvelles fonctionnalités pour de nouvelles applications (cf **Figure 1**).

1.1.2 Les contraintes d'un système ambiant : Hétérogénéité et Variation dynamique des entités de l'infrastructure

Les systèmes ambiants se caractérisent tout d'abord par la mise en œuvre de dispositifs et d'objets de la vie courante. Ces objets et dispositifs étant aussi différents que nombreux, nous parlons de *l'hétérogénéité sémantique* des objets et dispositifs d'un système ambiant, au-delà même de l'hétérogénéité technologique qui demeure une première difficulté à surmonter pour leur permettre d'interagir.

Par ailleurs, dans la plupart des systèmes ambiants, en dehors des quelques cas particuliers de systèmes mixtes statiques [Nigay03], les infrastructures logicielles évoluent dynamiquement sous l'impulsion des apparitions et disparitions des objets et dispositifs. Ces évolutions peuvent être dues à la mobilité de ces objets et dispositifs, mais aussi à des mesures d'économies d'énergie ou encore à des pannes. Il est alors primordial, dans ces environnements, de considérer les multiples entités hétérogènes, qui utilisent potentiellement divers protocoles de communication et qui peuvent apparaître et disparaître. Nous parlons alors de la *variation dynamique de l'infrastructure* des systèmes informatisés ambiants.

Illustrons les caractéristiques des systèmes ambiants avec la situation 6 du scénario prospectif livré en tâche 1. Bob, notre utilisateur représentatif, est alors équipé d'un WeCO (Wearable Computer) et se trouve dans un couloir d'hôtel : *« Bob se dirige vers sa chambre. MGenius perçoit cette action et règle d'ores et déjà le confort de la chambre. Bob, à une dizaine de mètres de sa chambre, constate qu'une porte s'éclaire, lui facilitant ainsi le repérage. Cependant, une autre personne le suit à quelques pas, et c'est en entendant son WeCo « bipper » devant la porte de sa chambre qu'il a la confirmation qu'il s'agit bien de la sienne. La porte s'ouvre automatiquement grâce à la clef électronique récupérée à l'arrivée. Bob entre. »* Les entités en jeu dans ce système ambiant sont les utilisateurs Bob et un inconnu (ils sont évidemment mobiles), mais aussi l'ensemble des dispositifs de la chambre de Bob, sa porte ainsi que son *wearable computer* (WeCo). La mobilité de l'utilisateur implique la nécessité de découvrir d'autres entités et de faire évoluer MGenius en fonction de ces entités. Il s'agit alors d'activer dynamiquement des services en fonction de la localisation intra-muros de l'utilisateur (couloir des chambres), de son objectif (trouver sa chambre) ainsi que de son environnement (pas seul).

Ces deux contraintes d'hétérogénéité et de dynamicité sont au cœur des contributions de CONTINUUM et plus particulièrement de la prise en compte du contexte pour l'adaptation logicielle. L'informatique ambiante doit être capable de s'adapter dynamiquement à tout changement, et donc de s'adapter au **contexte.** Le contexte apparaît alors, du point de vue informatique, comme le complément de l'ensemble des entités logicielles modifiables. Il est composé de :

- L'ensemble des infrastructures matérielles / logicielles
- L'environnement

- L'ensemble des entités ambiantes non informatisées comme les utilisateurs

1.2 Contexte et sensibilité au contexte

Le contexte n'est pas un concept nouveau en informatique. Dès les années soixante, les chercheurs en systèmes d'exploitation, théorie des langages et Intelligence Artificielle exploitaient déjà cette notion. Avec l'émergence de l'informatique ambiante, le terme est redécouvert et placé au cœur des travaux sans pour autant faire l'objet d'une définition consensuelle claire et définitive.

Toutefois, l'analyse de l'état de l'art conduit à ce double constat :

- Il n'y a pas de contexte sans contexte [Brézillon 02]. Autrement dit, le contexte n'existe pas en tant que tel. Il émerge, ou se définit, pour une finalité (ou utilité) précise. Dans notre cas, il s'agira d'adapter les applications aux évolutions du monde qui les entourent.
- Le contexte est un ensemble d'informations. Cet ensemble est structuré, il est partagé, il évolue et sert l'interprétation [Winograd 01]. La nature des informations, de même, l'interprétation qui en est faite, dépendent de la finalité.

1.2.1 Etymologie

Au sens étymologique du terme, le **Contexte** est issu du latin *contextus* "assemblage", *contextere* "tisser avec". Ainsi, les origines latines du terme dénotent l'aspect constructif (assemblage) et associatif (tisser avec) de la notion de contexte.

Parmi les dictionnaires de référence, citons :

- **Le Petit Robert** : "ensemble du texte qui entoure un élément de la langue (mot, phrase, fragment d'énoncé) et dont dépend son sens, sa valeur " ou encore " ensemble des circonstances dans lesquelles s'insère un fait".
- **L'encyclopédie Larousse** : "ensemble des conditions naturelles, sociales, culturelles dans lesquelles se situe un énoncé, un discours" [Encyclopédie Larousse]. Ou encore : "ensemble des circonstances dans lesquelles se produit un événement, se situe une action" [Encyclopédie Larousse].
- **Hachette Multimédia** : "ensemble des éléments qui entourent un fait et permettent de le comprendre" [Hachette Multimédia].
- **Le Grand dictionnaire numérique en ligne** : "Énoncé dans lequel figure le terme étudié" ou encore "Ensemble d'un texte précédant ou suivant un mot, une phrase, un passage qui éclaire particulièrement la pensée d'un auteur". Et, si l'on parle d'informatique [GrandDictionnaire] :"Ensemble d'informations concernant l'action du stylet, en rapport principalement avec sa localisation à l'écran, qui permet au système d'exploitation de l'ordinateur à stylet de différencier les commandes et l'entrée des données, et de fonctionner en conséquence."

Ces définitions partagent l'idée d'ensemble d'informations associé à quelque chose : "*ensemble […] qui entoure*", "*ensemble dans lequel se situe …*". La nature du "*quelque chose*" (texte, fait, énoncé, discours, événement, action, etc.), dépend précisément de l'utilité du contexte, confirmant le fait qu'il n'y a pas de contexte sans contexte. Et cette utilité, permettre de "*comprendre*", de "*fonctionner en conséquence*", ou de donner un "*sens, [une] valeur…*", peut s'exprimer de manière générique par "*servir l'interprétation*".

En somme, le constat énoncé précédemment, est cohérent avec les définitions générales des dictionnaires. Mais comment le contexte est-il abordé en informatique ambiante ?

1.2.2 Contexte et informatique ambiante

Schilit et Theimer introduisent l'expression *context aware* pour désigner un système doté d'un modèle du contexte. Le contexte, selon Schilit, inclut la localisation et l'identité des personnes et des objets à proximité ainsi que les modifications pouvant intervenir sur ces objets [Schilit 94a], [Schilit 94b]. Étudier le contexte, c'est répondre aux questions "*Où es-tu ?*", "*Avec qui es-tu ?*", "*De quelles ressources disposes-tu à proximité ?*". Il définit donc le contexte comme les changements de l'environnement physique, de l'utilisateur et des ressources de calcul. Ces idées sont reprises par Pascoe [Pascoe 98] puis par Dey [Dey 99]. Quant à lui, Brown restreint le contexte aux éléments de l'environnement de l'utilisateur [Brown 96], puis il introduit l'heure, la saison, la température, l'identité et la localisation de l'utilisateur [Brown 97].

Parallèlement aux travaux de Brown, des définitions émergent avec l'introduction explicite du temps et la notion d'état. Ryan assimile le contexte à l'environnement, l'identité et la localisation de l'utilisateur ainsi que le temps [Ryan 97]. Ward voit le contexte comme les états des environnements possibles de l'application [Ward 97].

En 1998, Pascoe définit le contexte comme *n* sous-ensembles d'états physiques et conceptuels ayant un intérêt pour une entité particulière [Pascoe 98]. Nous relevons ici la référence à la notion de pertinence.

Puis Dey précise la nature des entités en question: *« Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves. »* [Dey 99].

Dans leur étude sur la plasticité des IHM, Thevenin et al. aboutissent à une définition assez proche avec la notion de contexte d'interaction. Le contexte d'interaction est un triplet *<plate-forme - environnement - utilisateur>* où l'environnement est l'ensemble des entités (objets, personnes et événements) périphériques à la tâche courante et pouvant avoir un impact sur le comportement du système ou de l'utilisateur [Thevenin 99]. Nous relevons ici l'idée de tâche.

Après avoir détaillé trois catégories regroupant les différentes approches sur le contexte (les théories positivistes, phénoménologiques et critiques), Dourish, dans [Dourish 01], conçoit le contexte comme une forme d'information. Il le définit comme stable et définissable et insiste sur le fait que le contexte et l'activité (de l'utilisateur) sont deux éléments séparés. Pour lui, l'activité a lieu à l'intérieur du contexte.

Chen et Kotz [Chen00] posent la définition suivante : « *Le contexte est un ensemble d'états et de paramètres qui soit détermine le comportement d'une application ou bien dans lequel un événement d'application se produit et est intéressant pour l'utilisateur.* ». Dans cette définition, on voit

apparaître la notion d'informations passives qui décrivent le comportement à avoir et les informations actives qui déclenchent ce changement de comportement.

Enfin, les termes environnement et situation sont souvent utilisés comme synonymes de contexte, démontrant une absence de rigueur ou de maîtrise de ces notions.

En conclusion, le contexte est une notion obligatoirement définie par rapport à une ou plusieurs entités auxquelles il se rapporte. Après avoir défini un système ambiant, le contexte doit donc s'établir à partir du choix des entités privilégiées ou de références. Ainsi, si nous choisissons Bob, un utilisateur comme entité de référence, le contexte de Bob est l'ensemble des entités du système appelées à interagir avec Bob. Si nous considérons une entité informatisée, son contexte sera alors composé de son infrastructure et de l'environnement dans lequel elle évolue, lui même composé des autres systèmes et entités dont les utilisateurs.

Dans la suite de ce document, nous décidons d'étudier la sensibilité au contexte dans CONTINUUM, c'est-à-dire les procédés de prise en compte du contexte pour l'adaptation, selon trois axes d'étude :

- Un premier axe concerne le modèle du contexte choisi pour CONTINUUM.
- Un second axe sur la décomposition fonctionnelle des différentes étapes de traitement des informations pour la prise en compte du contexte.
- Un troisième axe, plus original, étudie le comportement dynamique des mécanismes de prise en compte du contexte au regard de l'évolution des éléments qui le composent comme de l'évolution des éléments de l'application.

Une fois ces différents axes expliqués, nous présenterons la manière dont le contexte sera pris en compte dans le projet CONTINUUM puis nous détaillerons l'architecture globale qui sera mise en œuvre dans le projet avant de conclure.

2 Le modèle du contexte de CONTINUUM

Bien que les différentes équipes de recherche académiques du projet aient une large expérience sur la modélisation du contexte, aucun des modèles conçus par ces équipes, présentés en annexe, ne couvre l'intégralité des problématiques de la continuité de service du projet CONTINUUM.

En effet, dans le cadre du projet de CONTINUUM nous devrons être capables :

- De faciliter la gestion de l'hétérogénéité tant technologique que sémantique des applications et des dispositifs : il s'agit de permettre aux applications d'utiliser l'ensemble des dispositifs présents en fonction de leurs besoins.
- De prendre en compte la problématique du passage à l'échelle : il convient de raisonner sur des préoccupations différentes de manières indépendantes et donc de ne pas travailler sur une notion de contexte « global » mais sur une combinaison de contextes conçus et développés indépendamment.
- De permettre la variabilité de l'environnement à l'exécution (runtime) : il faut pouvoir décider, à l'exécution, si une entité doit être prise en compte ou non et d'adapter l'application en fonction des entités présentes.
- De garantir la réactivité (latence) des mécanismes d'adaptation : l'application doit être adaptée le plus rapidement possible pour que celle-ci reste en conformité avec l'évolution de l'environnement.

Les travaux de l'équipe IIHM du LIG, [Rey05], consistent en une modélisation du contexte d'interaction basé les notions de rôles et de relations entre les entités. S'ils permettent de raisonner sur la détection de tel ou tel contexte durant la phase de design, ils ne permettent pas d'identifier au runtime si une entité doit être prise en compte ou non dans l'identification du contexte courant. De plus, ces travaux considèrent qu'il n'y a qu'un contexte à un instant donné qui correspond à l'état du monde restreint à ce qui a été défini au design.

Les travaux de l'équipe Rainbow de l'I3S, [Lavirotte05], portent sur la notion de zone contextuelle. Ces zones permettent de définir si une entité est ou n'est pas dans le contexte d'une autre entité. L'intérêt de ces travaux est l'évaluation locale asymétrique, et donc fortement décentralisée, des fonctions d'appartenances qui offre une bonne base pour le passage à l'échelle. Cependant, ce modèle ne facilite pas le raisonnement sur le contexte.

Les travaux de l'équipe HADAS du LIG, [Benyelloul10], se concentrent sur une modélisation à l'exécution de l'environnement de manière à raisonner sur ce modèle pour identifier le contexte courant. Le point fort de cette approche est son aspect déclaratif et sa capacité à travailler à un niveau d'abstraction plus élevé que les deux modèles précédents et donc de permettre de résoudre l'hétérogénéité sémantique que pose la continuité de service. L'inconvénient de cette approche est sa vision base de connaissance qui complexifie sa mise en œuvre en environnement non centralisé.

Le modèle de contexte adopté dans le projet CONTINUUM est donc une évolution du modèle de Crowley (lui-même très proche du modèle de Rey [Rey05]). Il reprend également des éléments

d'autres modèles des différents partenaires du projet (voir l'état de l'art en annexe de ce livrable) de manière à couvrir l'ensemble des problématiques liées à la continuité de services dans CONTINUUM.

De plus, nous nous sommes attachés à construire un modèle qui soit utilisable en phase de conception (*design time*) comme à l'exécution (*runtime*) :

- **Au design time**, le modèle du contexte doit aider les concepteurs d'applications à prévoir les contextes auxquels leurs applications devront être capables de s'adapter,
- **Au runtime**, le modèle du contexte doit fournir des mécanismes efficaces pour détecter les changements de contexte dans le but d'adapter l'application à ces changements.

2.1 Vocabulaire

2.1.1 Dispositifs et Observables

Comme nous l'avons présenté en introduction, un système ambiant repose sur un ensemble de dispositifs hétérogènes. Parmi ces dispositifs, certains, appelés capteurs, sont capables de fournir des données sur le système ambiant lui-même. De manière similaire aux définitions du contexte de Rey [Rey05] et de [Crowley02], nous appelons « observables » les données ainsi fournies.

Un* observable *est une donnée qui pourra être captée par des* dispositifs *de mesure (capteurs matériels ou logiciels) ou calculée à partir d'autres données captées.

2.1.2 Attributs et entités

Les observables sont des valeurs qui permettent de décrire les attributs des entités du monde (utilisateur, objet physique, …). Ces observables/attributs servent à caractériser (au design time) les différentes entités du monde et à les identifier (au runtime).

Une* entité *est un élément du monde décrit à l'aide d'*attributs *et qui correspond à un regroupement d'observables.

Aucune classification précise des entités n'est définie puisque celle-ci pourra varier en fonction des préoccupations (c.-à-d. en fonction des finalités visées). Pour cette raison, nous prévoyons de faire « cohabiter » plusieurs classifications (sous forme d'ontologies) en fonction des entités et des finalités prévues. Cela permet à chaque designer d'application de raisonner (au design time) sur sa propre ontologie, c'est-à-dire celle de son cœur de métier. Nous verrons plus loin dans ce livrable, ainsi que dans les livrables de la tâche 3, comment cette « cohabitation » entre les différentes ontologies est mise en œuvre au runtime.

2.1.3 Prédicats, Situations et Contextes

De manière à identifier la situation courante, nous définissons un ensemble de fonctions portant sur les entités du monde. Nous nommons ces fonctions des prédicats.

Les* prédicats *sont les rôles et les relations (spatiales, temporelles, ou autres) qui peuvent ou non être vérifiés par des entités.

Une* situation *est définie par un ensemble de prédicats vérifiés.

Nous appelons* contexte*, un ensemble de situations partageant le même ensemble de prédicats et d'entités et défini pour une même préoccupation.

2.2 Définition

De manière plus formelle, nous définissons le contexte de la manière suivante :

2.2.1 Contexte

Pour chaque préoccupation, le designer d'application définit deux ensembles (celui des prédicats et celui des entités) qui vont lui permettre de définir un contexte.

Un contexte ***C***, est défini sur deux ensembles ***C =*** (***Entités, Prédicats***) :

- ***Entités*** désigne l'ensemble des entités considérées par le designer de l'application. On ne retiendra que les entités pertinentes pouvant influencer l'application en fonction du contexte courant donc de la préoccupation considérée. ***Entités*** $= \{e_1, e_2, ..., e_n\}$ où e_i est une entité.
- ***Prédicats*** correspond à l'ensemble des prédicats mettant en jeu une partie des entités de l'ensemble ***Entités***. ***Prédicats*** $= \{p_1, p_2, ..., p_n\}$ où p_i est un prédicat. Quel que soit le prédicat p_i appartenant à l'ensemble ***Prédicats***, alors les objets de p_i appartiennent à l'ensemble ***Entités.*** p_i peut être d'arité 1 à n (avec n un entier positif).

Chaque contexte peut alors se décrire comme un graphe, où chaque nœud du graphe correspond à une situation.

Situation

Pour un contexte ***C =*** (***Entités, Prédicats***) donné, une situation S_i est défini par le couple $(P_i$, ***Entités)*** où :

- P_i est l'ensemble des prédicats vérifiés par les entités de l'ensemble ***Entités***. P_i *est donc une partie de l'ensemble* ***Prédicats*** $(P_i \subseteq$ ***Prédicats)***. De plus, quel que soit $p \in P_i$, il existe au moins n entités $e_i \in$ ***Entités*** tel que $p(e_1,...,e_n)$ soit vérifié (avec n l'arité de ***p***).

2.2.1.1 Changement de Situation

Il y a changement de situation pour un contexte ***C***, si l'ensemble P_i des prédicats vérifiés par les entités de l'ensemble ***Entités*** change. C'est-à-dire, si :

- Un prédicat de l'ensemble ***Prédicats*** n'est plus vérifié, ou si
- Un nouveau prédicat de l'ensemble ***Prédicats*** est vérifié.

2.2.2 Multiplicité des contextes

Comme nous l'avons indiqué, le designer définit un contexte C_p par préoccupation (exemple : contexte d'« économie d'énergie », de « sécurité », de « confort », …). Pour chaque contexte, il est donc nécessaire de définir les ensemble ***Entités***$_p$ et ***Prédicats***$_p$ associés à ce contexte : $C_p =$ (***Entités***$_p$, ***Prédicats***$_p$). L'intérêt d'une telle séparation est multiple. Il permet :

1. de diminuer le nombre des situations à modéliser dans chaque contexte (puisque le nombre de prédicats de chaque contexte est plus faible).

2. à chaque designer d'application de se concentrer sur son cœur de métier. Un spécialiste en sécurité ne s'occupera que de modéliser le contexte « sécurité » et les situations qui lui sont associées sans avoir à s'occuper d'autres préoccupations.
3. de prendre en compte de nouvelles préoccupations sans avoir besoin de modifier celles déjà définies.

2.2.3 Illustration de la méthodologie sur le scénario industriel

Nous montrons ici comment appliquer notre modèle de contexte au moyen du scénario industriel défini dans la tâche 1. Nous avons choisi d'illustrer la projection du modèle sur la partie du scénario industriel qui fera l'objet des démonstrateurs et des expérimentations en fin de projet. Cette partie commence à la section « itinéraire » page 24 du livrable 1.1 (scénario prospectif) puis couvre les parties « repérage », « première vanne » et « deuxième vannes » jusqu'à la page 26.

La première étape consiste à définir l'ensemble des contextes et donc des préoccupations de cette partie du scénario. Nous avons décidé d'en illustrer 4. Bien entendu, il est possible d'ajouter de nouvelles préoccupations ou d'en supprimer puisque celles-ci sont indépendantes les unes des autres.

Contexte 1 : Guidage routier

Entités :

Personne : Le fontainier que le système doit guider sur son lieu d'intervention.

Lieu : Le lieu de l'intervention.

Prédicats **:**

(1) *connaitLaDestination (Personne p, Lieu l)* : vérifié si la *Personne p* connaît le *Lieu* l d'intervention.

Situations **:**

Dans ce contexte, il y a deux situations possibles : soit Maurice (notre fontainier) ne connaît pas le *Lieu l* de destination et le système de guidage se mettra en route pour orienter au mieux cette personne. Soit notre fontainier connaît bien la destination, et le système ne se mettra pas en route.

Contexte 2 : MiseAJourPositionVanne

Entités :

Personne : Le fontainier qui porte les dispositifs de localisation et de communication (le WeCo).

Vanne : la vanne sur laquelle doit intervenir le fontainier.

Prédicats **:**

(1) *estProcheVanne (Personne p, Vanne v)* : vérifié si la *Personne p* est proche de la *Vanne* v. On considère que le fontainier est proche de la vanne lorsque, par exemple, il lit le tag RFID de la vanne avec son lecteur sans contact.

(2) *doitEtreVerifie(Vanne v)* : vérifié si la position de la *Vanne v* a besoin d'être mise à jour. On estime que les vannes ne doivent pas être mises à jour trop fréquemment (ce qui est actuellement le cas).

Situations **:**

Dans ce contexte, il y a 4 situations. Cependant, si une vanne ne doit pas être vérifiée, on peut ignorer le résultat du prédicat *estProcheVanne*, nous ramenant à 3 situations possibles. La première correspond à une vanne qui ne doit pas être vérifiée (le **Prédicat (2)** n'est pas vérifié). Dans la deuxième, (**Prédicat (2)** vrai, mais **Prédicat (1)** faux), le système attend que le fontainier soit proche de la vanne pour mettre à jour la position de la vanne. Pour la troisième situation (**Prédicats (1)** et **(2)** vrais), le système déclenche une mise à jour de la position de la vanne.

Contexte 3 : MiseAJourManœuvreVanne

Entités :

Personne : Le fontainier qui porte les dispositifs de localisation et de communication (le WeCo).

Vanne : la vanne sur laquelle doit intervenir le fontainier.

Prédicats **:**

(1) *estProcheVanne (Personne p, Vanne v)* : vérifié si la *Personne p* est proche de la *Vanne* v. On considère que la proximité du fontainier et de la vanne est fonction de leur position GPS respective.

(2) *doitEtreVerifie(Vanne v)* : vérifié si la *Vanne v* a besoin d'être mise à jour. On estime que les vannes ne doivent pas être mises à jour trop fréquemment (ce qui est actuellement le cas),

Situations **:**

Comme pour le contexte précédent, il y a 4 situations possibles. Cependant, si une vanne ne doit pas être vérifiée, on peut ignorer le résultat du prédicat *estProcheVanne* nous ramenant à 3 situations possibles. La première correspond à une vanne qui ne doit pas être vérifiée (le **Prédicat (2)** n'est pas vérifié). Dans la deuxième, (**Prédicat (2)** vrai, mais **Prédicat (1)** faux) le système attend que le fontainier soit proche de la vanne avant de déclencher le test de « manœuvre ». Pour la troisième situation (**Prédicats (1)** et **(2)** vrai), le système déclenche la procédure du test de manœuvre en commençant par en avertir le fontainier.

Contexte 3 : GestionTâches

Entités :

Personne : Le fontainier qui porte les dispositifs de localisation et de communication (le WeCo).

Prédicats **:**

(1) *estFiniTâcheCourante (Personne p)* : vérifié si la *Personne p* vient de finir sa tâche courante.

Situations **:**

Dans ce contexte il n'y a que deux situations. Dans la première, le fontainier est en train d'effectuer sa tâche. Dans la deuxième, qui est une situation transitoire, le fontainier vient de finir sa tâche (par exemple, il vient de finir le marquage et les tests sur la première vanne de sa préparation d'arrêt

d'eau). Le système met alors la tâche courante du fontainier à jour avec la vanne suivante, puis il attend la fin de cette nouvelle tâche.

2.2.4 Illustration de la méthodologie sur le scénario prospectif

Examinons maintenant le cas décrit dans le « moment 1 » du scénario prospectif. Celui-ci correspond à plusieurs préoccupations distinctes et donc à plusieurs contextes. Nous pourrions par exemple décrire 6 contextes différents pour couvrir l'ensemble des cas présentés dans ce « moment 1 ».

Contexte 1 : Réveil

Entités :

Personne : individu identifié comme tel par mGenius.

DispositifAlarmeSonore : dispositif pouvant émettre une alarme sonore et identifié comme tel par mGenius.

DispositifAlarmeLumineuse : dispositif pouvant émettre une alarme lumineuse et identifié comme tel par mGenius.

Prédicats **:**

(1) *estHeureReveil (Personne p)* : vérifié s'il est l'heure pour la *Personne p* de se réveiller.

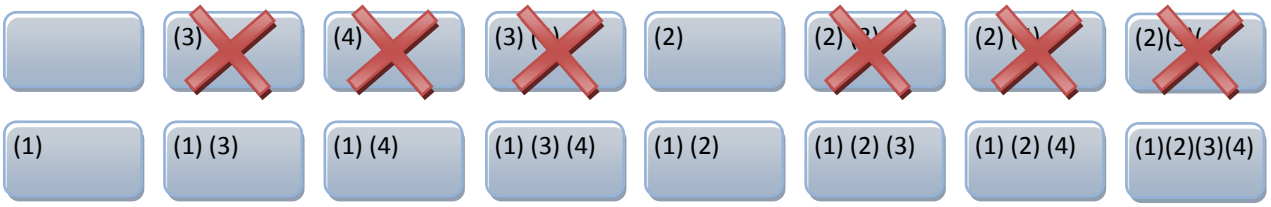
(2) *estRéveillé (Personne p)* : vérifié si la *Personne p* est réveillée.

(3) *alarmeSonoreOn (DispositifAlarmeSonore[] d)* : vérifié si au moins un *DispositifAlarmeSonore d* est actif.

(4) *alarmeLumineuseOn (DispositifAlarmeLumineuse[] d)* : vérifié si au moins un *DispositifAlarmeLumineuse d* est actif.

Situations **:**

A l'aide de ces 4 prédicats, nous obtenons 2^4 = 16 situations différentes représentées par le schéma ci-dessous. Cependant, le schéma peut être simplifié en supprimant les situations impossibles. Dans notre cas, les alarmes ne se déclenchant qu'à l'heure du réveil, les ***Prédicats (3)*** et ***(4)*** ne peuvent être vérifiés que si le ***Prédicat (1)*** est vérifié.



Contexte 2 : Petit-déjeuner

Entités :

Personne : individu identifié comme tel par mGenius.

Prédicats **:**

(1) *déjeune (Personne p)* : vérifié si la *Personne p* déjeune.

Situations **:**

Dans ce contexte, il y a seulement 2 situations. Celle où la personne p déjeune et celle où elle ne déjeune pas. mGenius effectue les modifications de l'application dès qu'il détecte que la personne qu'il considère, est en train de déjeuner. Cela a pour effet d'activer les sources d'informations que cette personne a l'habitude de visionner et d'écouter le matin.

Contexte 3 : Douche

Entités :

Personne : individu identifié comme tel par mGenius.

Prédicats **:**

(1) *estSousLaDouche (Personne p)* : vérifié si la *Personne p* prend sa douche.

Situations **:**

Dans ce contexte, il y a 2 situations : celle où la personne p est sous la douche et celle où elle ne l'est pas. mGenius déploie les adaptations idoines dès qu'il détecte que la personne qu'il considère, est en train de prendre sa douche. Cela a, par exemple, pour effet d'activer le filtrage des appels téléphoniques reçus en les redirigeant vers la boîte vocale.

Contexte 4 : Sécurité

Entités :

Personne : individu identifié comme tel par mGenius.

Prédicats **:**

(1) *estHorsDeLaMaison (Personne p)* : vérifié si la *Personne p* est à l'extérieur de la maison.

(2) *intrusion (Personne[] p)* : vérifié si une intrusion d'au moins une *Personne p* est détectée dans la propriété.

Situations **:**

Dans ce contexte, il y a 4 situations possibles. A chacune d'elles est associé un ensemble d'aspects d'assemblage déployé lorsque la situation correspondante est détectée. Par exemple, dans la situation **(1)**, mGenius déploie des adaptations relatives à la mise en sécurité de la maison (fermeture des portes et des volets des fenêtres, vérification du gaz, …) alors que dans la situation **(1)(2)** mGenius déclenche des adaptations d'alertes (transmission de l'alerte à la société de surveillance ou alerte des forces de police, enregistrement et/ou transmission des vidéo de l'infraction en cours, …)

Contexte 5 : Gestion de l'arrosage

Entités :

Personne : individu identifié comme tel par mGenius.

Hygromètre : dispositif qui indique l'humidité de la pelouse.

Prédicats **:**

(1) *doitEtreArrose (Hygromètre h)* : vérifié si la pelouse de la maison doit être arrosée (en fonction de l'heure et de l'hydrométrie, …).

(2) *personneSurLaPelouse (Personne[] p)* : vérifié si au moins une *Personne p* est détectée sur la pelouse.

Situations **:**

Dans ce contexte, il y a 4 situations possibles. . A chacune d'elles est associé un ensemble d'aspects d'assemblage déployé lorsque la situation correspondante est détectée. Par exemple, dans la situation **(1)**, mGenius déploie des adaptations relatives à la mise en œuvre du système d'arrosage alors que dans la situation **(1)(2)** mGenius déclenche des adaptations d'alertes pour avertir les personnes sur la pelouse que le système d'arrosage va se mettre en route. Une fois les personnes hors de la pelouse, on se retrouve dans la situation **(1).** Notons que les deux autres situations (la situation **(1)** et la situation **(2)**) sont identiques dans le cas de la gestion de l'arrosage.

Contexte 6 : Gestion des pannes

Entités :

Dispositif : ensemble des dispositifs détectés par mGenius et dont il a la charge.

Prédicats **:**

(1) *estEnPanne (Dispositif d)* : vérifié si la *Dispositif d* est en panne.

Situations **:**

Dans ce contexte, il y a 2 situations possibles : celle où un Dispositif d est détecté comme ayant un problème de fonctionnement et celle où ce dispositif fonctionne correctement. mGenius déploie les adaptations associées à chacune de ces situations. Par exemple, si un dispositif est détecté comme étant en panne, mGenius alerte une société de réparation et supprime ce dispositif de la liste des dispositifs valides pour que les fonctions l'utilisant se reconfigurent à l'aide de dispositifs équivalents.

2.3 Conclusion

Nous appuyant sur l'expérience des trois équipes académiques du projet, nous avons proposé un modèle du contexte permettant de répondre aux contraintes de l'informatique ambiante identifiées en introduction. Nous avons ensuite illustré l'utilisation de ce modèle sur les scénarios industriels et prospectifs définis lors de la tâche 1 du projet.

Voyons maintenant comment mettre en œuvre ce modèle au niveau de notre infrastructure logicielle de prise en compte du contexte. Mais avant de détailler celle-ci, commençons par étudier les rôles et les modèles de telles infrastructures.

3 Décomposition fonctionnelle de la prise en compte du contexte

3.1 Du tout « laissez faire » au tout « transparent »

Les premières applications sensibles au contexte intégraient de manière ad-hoc un ensemble de mécanismes pour la prise en compte du contexte. Il s'agissait souvent d'une prise en compte limitée à certains types d'informations contextuelles (localisation, système …), ceci dû, entre autres, aux contraintes imposées par de tels mécanismes [Dey99]. Or, proposer des solutions à toutes ces contraintes impose bien souvent un investissement trop lourd dans le développement de l'application. Typiquement, les informations contextuelles sont obtenues de sources hétérogènes et distribuées. De plus, elles ne peuvent généralement pas être utilisées telles que transmises par les capteurs et doivent subir un traitement pour être exploitables. La mise en place de mécanismes pour traiter ces problèmes prend alors une importance trop grande dans le cycle de développement de l'application. D'autre part, ces premières approches restaient difficiles à maintenir et à faire évoluer. En effet, le code de cette préoccupation de prise en compte du contexte était alors fortement enchevêtré dans celui des préoccupations fonctionnelles de l'application. Ces approches sont dites « laissez faire » [Saty96].

Dans une approche traditionnelle de génie logiciel, les mécanismes de prise en compte du contexte sont donc de plus en plus externalisés des applications, ceci pour offrir des solutions réutilisables et pour simplifier la conception d'applications sensibles au contexte [Dey99]. Si les applications ont toujours la possibilité de disposer de leurs mécanismes ad-hoc de prise en compte du contexte, elles privilégient un travail avec une couche logicielle générique servant d'interface avec l'environnement de l'application.

Les intergiciels sont des couches logicielles qui peuvent offrir de telles possibilités. Issarny et al en donnent la définition suivante: « *Middleware is a software layer that stands between the networked operating system and the application and provides well known reusable solutions to frequently encountered problems like heterogeneity, interoperability, security, dependability*. »[Issarny07]. Mais la principale caractéristique des intergiciels est leur capacité à traiter avec de multiples clients hétérogènes. Ils offrent donc aux applications une vue unifiée des systèmes avec lesquels elles interagissent. Dans le domaine de l'informatique ambiante, la prise en compte du contexte repose essentiellement sur une multitude de dispositifs hétérogènes et potentiellement mobiles. Les intergiciels sont donc une approche privilégiée pour offrir aux applications les mécanismes de prise en compte du contexte.

Les premiers intergiciels de gestion du contexte tels que la Context Toolkit [Dey99], la Context fabric [Hong02], SOCAM [Gu05], les Contexteurs [Rey05] ou encore CoWSAMI [Issarny08], offraient des mécanismes de collecte, de stockage mais aussi de traitement des informations captées, proposant ainsi une factorisation de l'observation du contexte. Ils incluent, dans des approches plus ou moins décentralisées, des mécanismes de découverte d'entités hétérogènes mais aussi de déduction et de filtrage des informations collectées. Mais l'observation du contexte restait alors une sous-

problématique à celle de la prise en compte du contexte. En effet, comme cela est montré dans SAFRAN [David06], l'adaptation du comportement de l'application est aussi une préoccupation transverse qui fait partie de la prise en compte du contexte [Bottaro07]. Ainsi, de plus en plus, les intergiciels englobent la totalité des fonctions de prise en compte du contexte, de son observation à l'adaptation du comportement de l'application. On parle d'approche « transparente » [Saty96], c'est-à-dire que cette préoccupation est transparente pour le développeur de l'application. Parmi ces intergiciels dédiés au contexte, citons Amigo [Sacchetti05], SAFRAN [David06], Camido [Behlouli06], Gaia [Roman02], MADAM [Floch06] ou encore RCSM [View02], [Bottero07].

3.2 Les grands blocs fonctionnels de la décomposition

Les mécanismes classiques de prise en compte du contexte reposent sur une décomposition fonctionnelle motivée par la réutilisabilité et l'évolutivité. Dans le cadre de la prise en compte du contexte, cette décomposition repose généralement sur les grandes fonctionnalités que l'on retrouve dans [Coutaz05]. Dans un premier temps, il s'agit de recueillir des données brutes. Il s'agit de la phase de collecte des informations contextuelles. On parle également de cette étape comme celle de la perception. Par la suite, ces données sont transformées en observables symboliques, en informations de plus haut niveau d'abstraction, par exemple à l'aide d'ontologies. On déduit, par exemple, la proximité entre deux entités (savoir si elles sont dans un même bureau) à l'aide de leur localisation et d'un plan du bâtiment. A la suite de quoi, ces données servent lors d'une phase de décision, d'identification de la situation, qui produit un plan de réaction. Ce dernier est alors mis exécuté par le mécanisme d'adaptation aussi appelé mécanisme de contrôle. Bien entendu, ces étapes peuvent être affinées comme le montre la **Figure 2**.

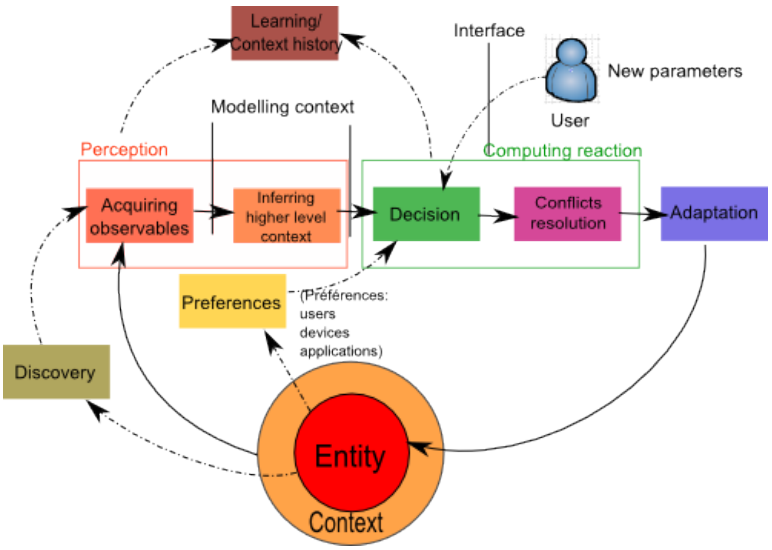


Figure 2. Le cycle de la prise en compte du contexte [Ferry08]

Dans ces architectures, il s'agit donc d'organiser entre elles un ensemble de fonctionnalités qui peuvent être implémentées sous diverses formes : services, composants, bibliothèques … Nous allons maintenant présenter plus en détails les grandes fonctionnalités de cette décomposition.

3.2.1 Observation/Capture

D'après Schilit [Schilit94b], la sensibilité au contexte est la capacité d'un programme ou d'un dispositif à percevoir les divers états de l'environnement et de lui-même. L'acquisition d'informations

relatives au contexte s'effectue à l'aide de sondes, de capteurs. Ces capteurs sont les interfaces entre l'intergiciel et/ou l'application, et son environnement.

Traditionnellement, lors de la réalisation de systèmes utilisant de tels mécanismes, les développeurs choisissent leurs sondes statiquement. La haute variabilité de l'infrastructure d'un espace ambiant rend cette solution de moins en moins acceptable puisqu'il devient impossible de définir à l'avance les capteurs qui seront à la disposition de l'application dès sa conception [Lei02]. Les mécanismes de découverte de nouveaux capteurs sont une première solution permettant d'adapter les mécanismes de capture du contexte en fonction de l'environnement. Cependant, la découverte peut rester statique, puisqu'il peut s'agir pour un « consumer » de demander à un « broker » comment trouver un service désiré. Il s'agit donc d'un mécanisme de requêtes dans un annuaire connu. Par exemple, dans SOLAR [Chen03] et Daidalos [Strimpakou05], les capteurs ou observateurs ne peuvent pas être découverts à partir d'une source inconnue. Ils doivent s'annoncer au gestionnaire de capteurs pour devenir une source d'information contextuelle dans le système. Ainsi, plus que la simple découverte, les notions d'apparition et de disparition de capteurs doivent être considérées. Elles mettent en jeu des mécanismes dynamiques qui ne peuvent pas être implémentés par des systèmes de requêtes mais à travers des annonces dynamiques (broadcasting). Prendre en considération l'apparition et la disparition des capteurs nécessite de ne pas avoir recours à un annuaire connu.

Les communications avec les capteurs découverts peuvent alors s'opérer de deux manières. La première approche impose à la partie logicielle de prise en compte du contexte d'aller requérir l'information auprès des capteurs et ce, à intervalle régulier. Le système impose alors sa propre dynamique pour la collecte d'informations contextuelles. Dans la seconde, les informations peuvent être fournies à la plate-forme logicielle en mode push, par exemple, sous la forme de remontées de messages à chaque modification de l'état de l'observable. Cette approche reste la plus respectueuse de la dynamique de l'environnement puisqu'elle n'introduit comme latence, pour la prise en compte de cette information, que celle de l'envoi du message.

3.2.2 Transformation en observables

Les données captées ne sont parfois pas toujours suffisantes pour alimenter les raisonnements des couches logicielles supérieures. Ainsi, l'étape de transformation en observables peut incorporer des mécanismes d'interprétation des informations captées dans l'optique d'en déduire des données de plus haut niveau ou encore d'améliorer la qualité des informations collectées. Il peut alors s'agir d'agréger (fusionner) ou encore de filtrer les informations remontées par les capteurs mais aussi de produire un raisonnement sur ces informations, par exemple, sémantique à l'aide d'ontologies, afin d'obtenir des informations plus pertinentes au bon niveau d'abstraction.

3.2.3 Décision

Une fois collectées, les données relatives au contexte doivent être évaluées afin de calculer la réaction au changement de contexte.

Différents mécanismes comme les ontologies, les règles ECA (événement-condition-action) ou encore la logique floue, offrent de telles possibilités. Le rôle de ces mécanismes se borne souvent à déclencher une réaction de manière opportune. Pour ce faire, ils s'appuient généralement sur des approches événementielles ou utilisant des techniques d'inférence. SAFRAN [David06] propose par

exemple un mécanisme de sélection de règles d'adaptation ECA dont les événements peuvent être propres à l'application (événements « endogènes ») ou provenir de son environnement (événements « exogènes »). La complexité de ces mécanismes est fortement dépendante des capacités du système à collecter et à interpréter les informations contextuelles. Mais les mécanismes de décision peuvent aussi disposer d'interfaces homme-machine pour leur déclenchement et leur configuration par l'utilisateur. D'autre part, si les observables courants sont à la base de cette évaluation, la possession d'un historique des contextes observés et des actions effectuées peut entrer en compte dans le choix d'une adaptation pour une situation donnée. Un historique permet entre autre de raisonner sur le contexte courant dans le cas où les informations viendraient à manquer. De plus, l'étude de l'observable doit être couplée à celle des différentes préférences fournies par l'utilisateur et qui sont un complément d'informations. Enfin, l'utilisation de techniques d'apprentissage peut être également envisagée. De tels mécanismes permettraient, entre autre, de réaliser des adaptations proactives. La phase d'évaluation peut donc être vue comme la détection, la reconnaissance d'une situation déclenchant une adaptation de l'application.

3.2.4 Réaction

Les réactions aux changements qui interviennent dans le contexte d'une application ont pour but d'adapter son comportement. Et puisque l'environnement est en perpétuelle évolution, ces adaptations doivent être dynamiques.

On distingue deux techniques permettant de réaliser des adaptations dynamiques logicielles : l'adaptation paramétrée et la reconfiguration [McKinley04] aussi appelée adaptation compositionnelle.
La première consiste à modifier certaines variables qui influent sur le comportement de l'application. Si cette approche reste relativement simple à mettre en place, elle permet uniquement de paramétrer des algorithmes, composants, stratégies déjà existantes et non pas d'en incorporer de nouvelles. Cela nécessite de prévoir, lors de la conception, toutes les possibilités de modifications de l'application.
A l'inverse, la reconfiguration vise à substituer des algorithmes ou stratégie d'une application, par d'autres dans l'optique de coller au mieux à une situation donnée. Cette approche est plus évolutive et permet de proposer et d'intégrer de nouvelles adaptations dans un système à l'exécution. La capacité d'un système à procéder à une reconfiguration implique une certaine modularité. En effet, un système qui ne respecterait pas cette propriété devrait être entièrement changé. A l'inverse, s'il est conçu comme un assemblage de modules, il devient alors possible d'en remplacer uniquement certaines sections. Cette décomposition peut alors être obtenue à l'aide de différents paradigmes logiciels. Dans ce cadre McKinley [McKinley04] identifie la programmation orientée objet, la programmation par composants et la programmation par aspects.

3.2.5 Limitations des architectures verticales

Les décompositions fonctionnelles entraînent souvent la mise en place d'architectures dites « verticales », c'est-à-dire qui se composent d'un ensemble de couches superposant les diverses fonctionnalités. Les architectures de prise en compte du contexte ont donc un rôle central puisqu'elles interagissent à la fois avec l'environnement mais aussi avec les applications (**Figure 3**).

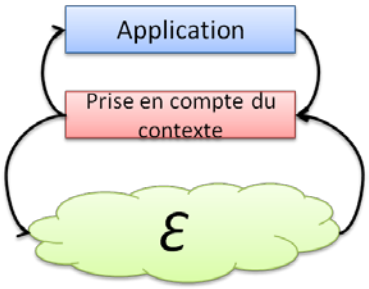


Figure 3. Le mécanisme de prise en compte du contexte suit deux dynamiques.

Une telle architecture doit alors respecter au mieux les dynamiques de l'application mais aussi celle de l'environnement de l'application. En effet, l'environnement évolue perpétuellement et le système doit répondre aux sollicitations de cet environnement mais aussi de l'application (et bien entendu de l'utilisateur). Ainsi, la réactivité du système à ces sollicitations est une préoccupation majeure en informatique ambiante, que ce soit pour son temps de réponse ou le déclenchement des adaptations. Nous considérons qu'une application se trouve toujours dans l'un des états présentés dans la **Figure 4** lors d'une adaptation. Les états 1 et 3 décrivent l'application avant et après son adaptation. Ce sont des états pour lesquels l'application est cohérente avec son environnement : elle est dans un état pertinent, c'est-à-dire que son comportement est celui attendu. Durant l'état 2, l'application est en phase d'adaptation et par conséquent souvent indisponible. L'application n'est alors plus dans un état cohérent avec son environnement.

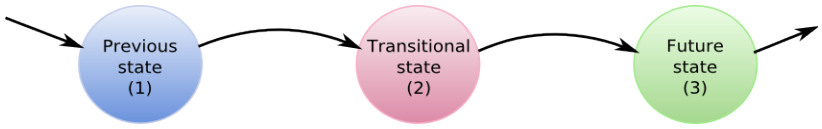


Figure 4. Les états d'une application adaptative.

Ainsi le temps passé dans l'état 2 doit correspondre aux différentes dynamiques exposées précédemment :
L'adaptation doit offrir une dynamique compatible avec la dynamique d'évolution de l'environnement de l'application [Ferry09].

- C'est-à-dire ne pas laisser l'application dans l'état (2) trop longtemps, état dans lequel elle n'est plus utilisable. Il faut donc limiter au maximum le temps d'adaptation pour obtenir rapidement une application fonctionnelle (3).
- Il ne faut pas non plus laisser trop longtemps l'application dans l'état (1) qui ne convient plus après une variation de l'environnement. L'application peut ne plus utiliser les fonctionnalités offertes par son infrastructure ou encore risque d'essayer d'en utiliser qui n'y sont plus.
- Tout cela au risque de voir sombrer l'application dans un état instable qui consisterait à ne jamais atteindre l'état suivant (3) avant une nouvelle évolution de l'environnement. Il est alors possible de tomber dans une forme d'indéterminisme, c'est-à-dire à ne plus connaître l'état de l'application ni comment elle va être adaptée.

Comme nous l'avons vu, l'adaptation doit également offrir une dynamique compatible avec le fonctionnement de l'application et/ou de l'utilisateur.

- c'est-à-dire de ne pas passer des états (1) à (3) trop fréquemment et produire une application incohérente et instable.

- A l'inverse vis-à-vis de l'utilisateur, une trop forte latence peut le perturber et le détourner du système [MacKenzie93]

Comme nous l'avons vu, la plupart des intergiciels repose sur des architectures verticales et se concentre plus particulièrement sur certaines fonctionnalités du cycle de prise en compte du contexte (cf. Figure 2). Ce type d'architecture apporte de nombreux avantages, particulièrement lors des phases de conception, et offre entre autre des facilités de réutilisabilité. S'il est aisé à mettre en place, il a pour principal inconvénient de toujours reposer sur des approches centralisées en au moins un des points du cycle de traitement. Or, dans le cadre de l'informatique ambiante, où les systèmes doivent considérer une multitude de données et gérer de nombreux dispositifs, la centralisation reste difficilement envisageable. En effet, cela mène souvent à des goulots d'étranglement aussi bien en terme de performances (donc de réactivité) que de robustesse.

En conséquence, ce type d'architecture ne permet pas de maîtriser les diverses dynamiques exposées précédemment puisque la réactivité du système à un changement de contexte dépend de la réactivité de chacune des parties de traitement. Elles sont donc toutes évaluées au même moment à partir des mêmes informations traitées. Cela mène à un système dont la dynamique est fixe. Lorsque des raisonnements complexes sont utilisés, comme dans SOCAM [Gu05], les changements peuvent être pris en compte en une seconde en moyenne, sans compter le temps d'adaptation. Ainsi, lorsqu'un grand nombre de changements intervient dans l'environnement, et qu'il faut adapter rapidement l'application, des temps de réaction plus courts sont nécessaires pour que cette dernière puisse être en phase avec l'environnement.

En conséquence, la complexité des représentations et des traitements des informations contextuelles impacte sur la dynamique de la prise en compte du contexte. Il est alors indispensable d'étudier une autre décomposition de la prise en compte du contexte selon ces dynamiques.

4 Pour la maîtrise des dynamiques de la prise en compte du contexte

4.1 Définition de la décomposition comportementale

Les architectures horizontales sont tirées du monde de la robotique et de l'intelligence artificielle. Il s'agit de spécialiser un cœur d'application minimal avec des composantes particulières (horizontales) [Zhang04]. Les composantes sont indépendantes les unes des autres et s'exécutent en parallèle. Chaque composante peut alors être connectée au monde via des capteurs et agir sur son environnement via des actionneurs. Ce type d'architecture introduit une nouvelle approche de décomposition : la décomposition comportementale.

D'après Bryson [Bryzon01], la décomposition comportementale est une approche architecturale qui décompose l'intelligence en termes de comportement tels que manger ou marcher, plutôt qu'en des processus génériques comme planifier ou observer. Un comportement correspond alors à une composante horizontale et des activités peuvent se composer d'ensembles de comportements managés. Les comportements peuvent donc avoir différents niveaux de complexité et ne pas être connectés au monde via des mécanismes de perception. Ainsi le comportement le plus simple sera sans perception ni état [Bryzon01] (par exemple un comportement pour un robot qui serait « rouler en continu»). Une contribution forte de ces approches est de ne pas voir le système comme une séquence de processus mais plutôt comme une parallélisation de processus et donc de comportements qui, ensemble, peuvent produire une activité cohérente. Il s'agit en réalité de décomposer le système en comportements (composantes horizontales) et ainsi de décomposer des comportements complexes en de plus simples avec une stratégie « *divide and conquer* ».

Ces architectures respectent donc les caractéristiques suivantes :

- Chaque composante capture directement dans l'environnement ce qui est pertinent pour elle de manière à ce qu'elles puissent mettre en œuvre leur comportement.
- Il n'y a pas de représentation globale de l'environnement pour une approche décentralisée : « l'environnement est lui-même sa meilleure représentation » [Brooks91]. Les seules données exactes sur l'environnement sont les données obtenues immédiatement par les capteurs.
- Ces architectures comprennent de nombreuses composantes horizontales de faible complexité pour une réactivité maximale.

Les architectures horizontales requièrent un mécanisme de coordination (un gestionnaire) afin de combiner les données générées par chaque composante pour obtenir un comportement final global cohérent. Dans les premières architectures horizontales de Brooks [Brooks91], les « subsumption architectures », il s'agit d'un mécanisme de subsumption[1], c'est-à-dire qu'un comportement de niveau N+1 peut subsumer le rôle des comportements inférieurs en supprimant leurs sorties. Plus le mécanisme est simple, plus la réactivité est élevée. A partir de ces caractéristiques, nous pouvons définir une activité comme un ensemble managé de comportements (**Figure 5**).

[1] Raisonnement par lequel on met une idée sous une idée plus générale.

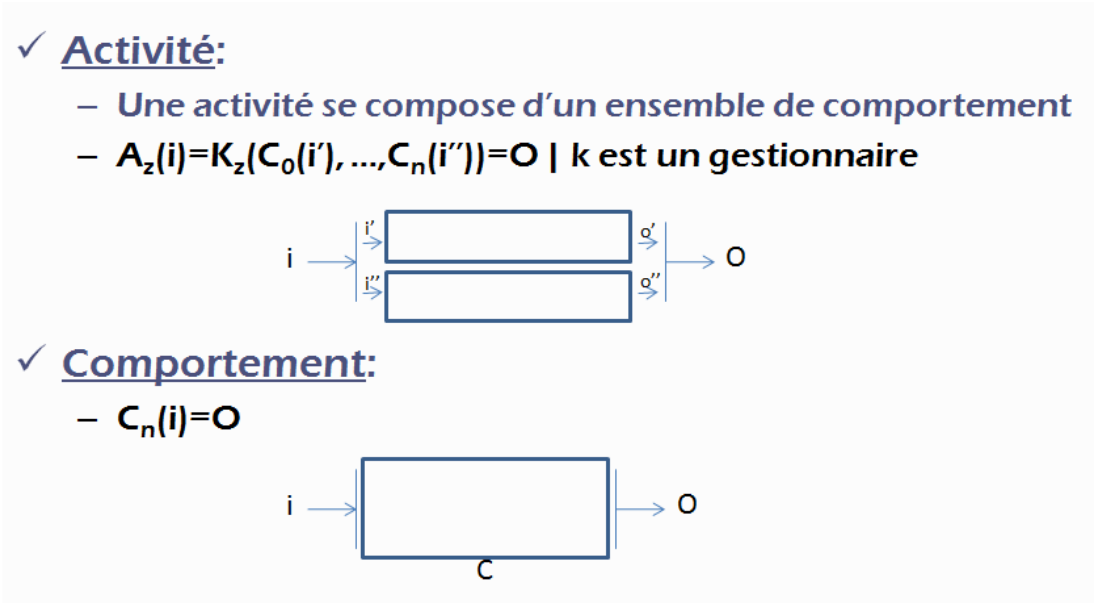


Figure 5. Définition d'une activité dans une approche comportementale.

Ce type d'architecture permet de mieux maîtriser les dynamiques de l'environnement mais aussi de l'application. En effet, le traitement réalisé par une composante horizontale n'est effectué que lorsque nécessaire. De plus, celui-ci est plus rapide puisque chaque composante ne traite que ce qui est pertinent pour elle. Les diverses composantes horizontales étant indépendantes les unes des autres, elles offrent leur propre dynamique qui n'est plus tributaire des autres traitements. Il devient alors possible d'écrire des composantes avec des niveaux de complexité variés afin de respecter, au mieux, les dynamiques imposées par l'environnement et l'application.

4.2 Décomposition comportementale dans les intergiciels de prise en compte du contexte

Une architecture horizontale peut être mise en place dans les intergiciels de prise en compte du contexte. Les comportements étant indépendants les uns des autres, chacun d'eux doit mettre en place un mécanisme de prise en compte du contexte pertinent pour le dit comportement. Comme nous l'avons vu précédemment, un comportement collecte des informations sur son environnement et, après traitement, réalise un certain nombre d'actions sur celui-ci. Dans le cadre de la prise en compte du contexte, un intergiciel reposant sur ce type d'architecture se compose d'un ensemble de comportements allant chacun de la collecte d'informations à des adaptations de l'application. L'ensemble de la préoccupation de prise en compte du contexte est nécessairement mis en place dans l'intergiciel.

Par conséquent, le mécanisme de management tardif des comportements devra résoudre des conflits d'adaptation. On parle alors de fusion tardive des adaptations traitée dans la sous tâche 2.3 du projet CONTINUUM.

5 Contexte et prise en compte du contexte dans CONTINUUM

5.1 Du fonctionnel dans du comportemental : une approche hybride

Comme nous l'avons vu précédemment, un comportement perçoit des informations de son environnement et agit sur celui-ci en retour. Un comportement identifiant une action à réaliser en fonction d'un contexte pertinent, respecte les grandes étapes que l'on retrouve dans les décompositions fonctionnelles, à savoir : perception, décision, réaction. Un tel comportement peut également être décomposé en fonctionnalités. Nous introduisons une approche hybride qui consiste en une décomposition fonctionnelle au sein d'une décomposition comportementale : un comportement se compose lui-même d'un ensemble de fonctionnalités qui dans le cadre de la prise en compte du contexte vont de l'observation aux réactions.

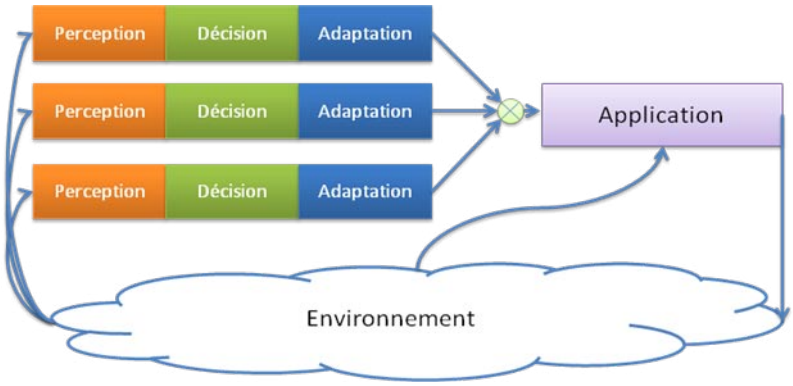


Figure 6. Approche hybride.

En conséquence, nous définissons un comportement comme l'illustre la **Figure 6**. Cette définition implique une inversion méthodologique dans la conception des mécanismes proposés par l'intergiciel. En effet, il ne s'agit pas de découper des comportements dans un ensemble de fonctionnalités, mais d'écrire un ensemble de comportements à partir de fonctionnalités réutilisables. Les grandes étapes de la conception d'architecture verticales sont :

1. Identification des fonctionnalités
2. Spécification du flot de fonctionnalités
3. Représentation des données traitées
4. Spécification des entrées/sorties
5. Réalisation

Les trois premières étapes, qui sont primordiales dans la conception d'architectures, reposant sur une décomposition fonctionnelle, perdent leur importance dans la conception d'architectures horizontales pour lesquelles la spécification des entrées/sorties d'un comportement sont un pré-requis pour la suite de sa conception. Les grandes étapes de la conception d'architectures horizontales sont :

1. Identification d'un comportement
2. Spécification des entrées/sorties

3. Identification des fonctionnalités
4. Spécification du flot de fonctionnalités
5. Réalisation

Lors de la création d'architectures horizontales, la spécification des entrées/sorties prend une importance particulière puisque, contrairement aux approches fonctionnelles classiques, l'agrégation des données ne se fait pas sur les entrées du système (d'où le besoin de représentation des données dans les approches fonctionnelles) mais bien entre les sorties des comportements.

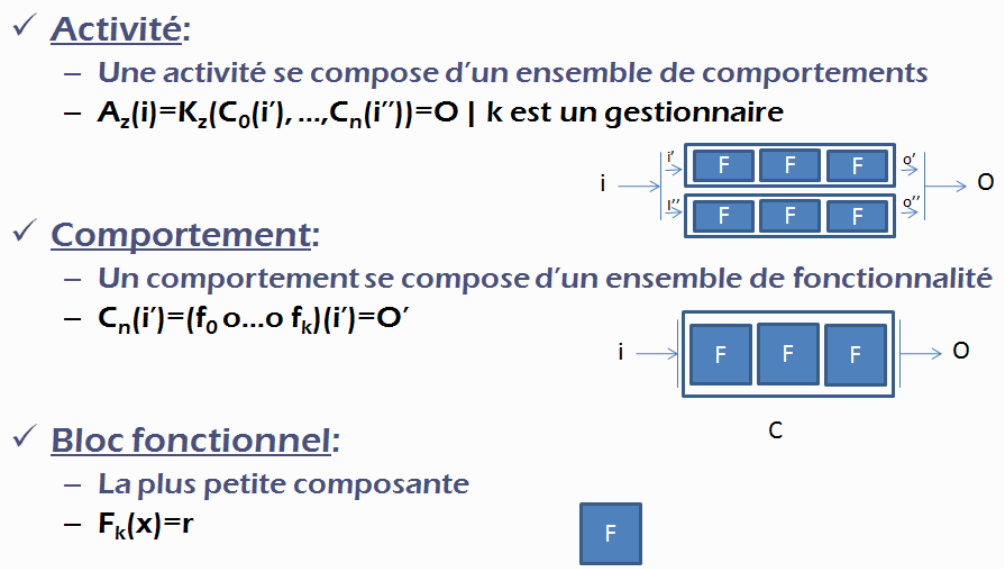


Figure 7. Définition d'une activité dans une approche hybride.

Les approches hybrides tirent partie des avantages des deux décompositions que sont en particulier les capacités de maîtrise des dynamiques et d'indépendance offertes par la décomposition horizontale et la forte réutilisabilité offerte par la décomposition fonctionnelle. Plus particulièrement, l'approche hybride offre par rapport à l'approche comportementale :

- une modularité dans les comportements et donc des facilités de maintenances mais aussi d'évolution d'un comportement,
- des facilités de réutilisabilité de ces modules et une meilleure séparation des préoccupations fonctionnelles d'un comportement.

5.2 Architecture de la plate-forme CONTINUUM

Notre approche pour la prise en compte du contexte dans l'architecture CONTINUUM repose sur un intergiciel qui s'appuie sur une décomposition hybride. Dans cette architecture, le mécanisme de prise en compte du contexte repose sur des compositions de services de l'infrastructure logicielle. Cette infrastructure logicielle sera mise en œuvre via WComp, une implémentation du modèle SLCA [Tigli09]. Le mécanisme de prise en compte du contexte repose, lui, sur une décomposition hybride sur trois niveaux. Dans les sections suivantes, nous présentons plus en détails l'infrastructure sur laquelle repose l'architecture de la plate-forme CONTINUUM puis les trois niveaux du mécanisme de prise en compte du contexte.

5.2.1 Infrastructure

Nous avons vu que les environnements ubiquitaires mettent en jeu de nombreux objets intelligents, communicants et potentiellement mobiles. Cette multiplicité implique une grande diversité d'objets

toujours plus petits et puissants comme des Smartphones, ordinateurs portables ou encore divers capteurs.

Les approches orientées services [MacKenzie93] permettent de répondre aux besoins de découverte et d'hétérogénéité technologique au travers, entre autre, de l'utilisation de contrats et de standards comme ceux du Web. S'il est essentiel de disposer dans de tels environnements de mécanismes de découverte, il faut aussi gérer l'apparition et la disparition de ces services. Pour cela, diverses solutions technologiques existent et notamment les services pour dispositifs.

Les services pour dispositifs sont une extension des services dits classiques avec deux contributions majeures : les communications asynchrones à l'aide d'événements et un mécanisme de découverte décentralisée qui permet de se dispenser d'annuaire de services. Ils répondent au besoin de prise en compte de l'apparition et de la disparition des entités composant l'infrastructure logicielle d'une application [Jung07]. Actuellement, seules deux implémentations des services pour dispositifs existent : UPnP[2] et DPWS[3]. La **Figure 8** présente le métamodèle d'UPnP.

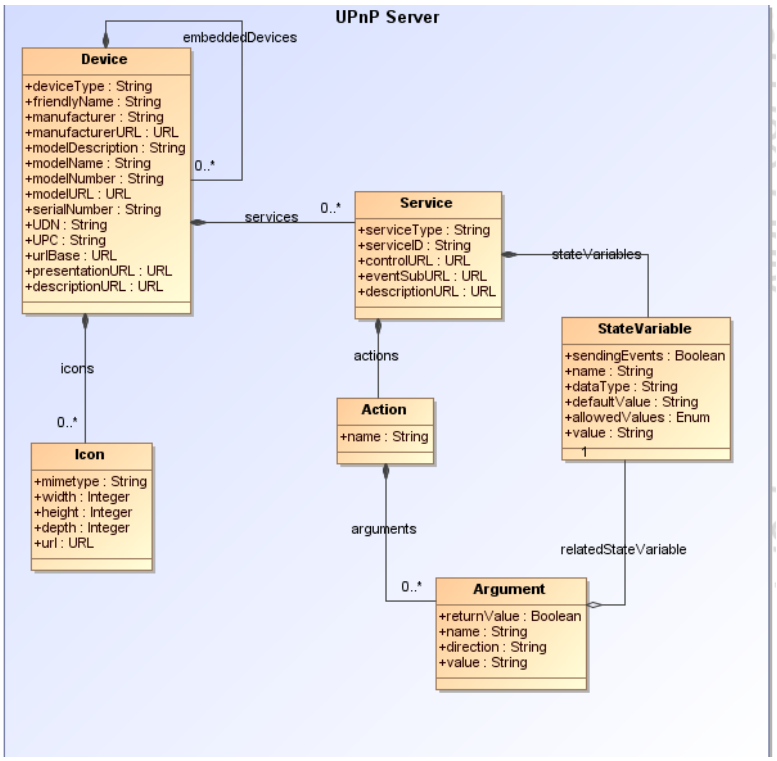


Figure 8. Métamodèle de UPnP.

Un système ubiquitaire se compose donc d'un ensemble de services pour dispositifs. Deux types de composition de services principaux existent : l'orchestration et la chorégraphie [Singh05]. L'orchestration nécessite une entité centralisée qui effectue tous les appels de méthodes sur les services de l'application et relaie les messages entre les services. La chorégraphie propose une approche décentralisée, mais casse en partie la propriété de boîte noire des services. En effet, la chorégraphie suppose que les services sont capables de s'organiser de façon autonome pour communiquer. Elle est plus complexe à mettre en œuvre, puisque les services doivent se connaître, et les adaptations éventuelles doivent être embarquées dans chaque service. L'orchestration semble

[2] Universal Plug and Play Forum: http://www.upnp.org/
[3] Device Profile for Web Services: http://www.ws4d.org/

donc plus adaptée à l'informatique ambiante, puisque les changements fréquents dans l'infrastructure seraient complexes à gérer dans tous les services [Cardoso09].

Les services pour dispositifs sont donc les solutions aux problématiques de découverte, de dynamicité, d'hétérogénéité mais aussi d'apparition/disparition de dispositifs. Pour créer des applications à partir de ces derniers nous utilisons une architecture de composition de services à base de composants légers : SLCA [Hourdin08]. SLCA offre la possibilité d'orchestrer et de composer dynamiquement des services pour dispositifs à l'aide d'assemblages de composants légers s'exécutant dans des containers. Des outils externes, les designers, assurent la génération dynamique de composants clients de web services pour dispositifs. Cette génération est réalisée grâce à l'utilisation des descriptions des services : les contrats. Nous appelons ces composants des composants proxy.

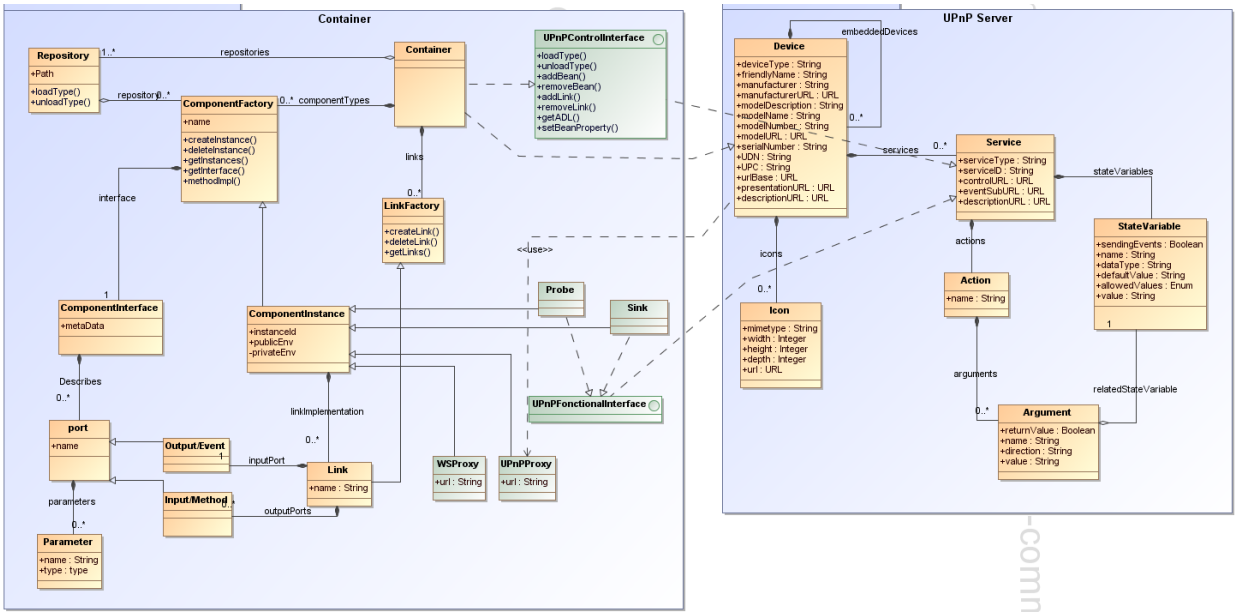


Figure 9. Métamodèle de SLCA. On retrouve à droite le métamodèle de UPnP et à gauche celui du container SLCA. Dans ce dernier, les couleurs permettent de délimite la partie locale (LCA) du modèle (en orange), de son extension au monde des services (en vert).

Comme le montre la **Figure 9**, les containers permettent de gérer des assemblages de composants à l'exécution. Les composants peuvent être chargés et déchargés dynamiquement. Ils peuvent être ajoutés, retirés ou connectés entre eux dynamiquement. Les composants proxy, en fonction de la présence de services pour dispositifs dans l'infrastructure du container, sont générés, chargés et instanciés dynamiquement de manière automatique. Cette génération est hautement réactive [Hourdin08]. Il y a donc une correspondance directe entre l'infrastructure sur laquelle repose l'application et l'assemblage de composants. L'ensemble des composants d'un container applicatif est donc le reflet des services pour dispositifs présent dans l'infrastructure. La plate-forme CONTINUUM offre ainsi les mécanismes nécessaires à la gestion de l'apparition/disparition des services pour dispositifs de manière à ce que ce reflet (cette cohérence) soit maintenu au cours du temps.

Une application peut donc être créée à partir d'une infrastructure logicielle sous forme d'assemblages de composants. Les composants proxy peuvent être combinés entre eux mais aussi avec des composants purement fonctionnels. Les communications entre ces divers composants sont

événementielles pour une meilleure dynamicité et un couplage faible entre composants. Un container peut lui-même être exporté comme un service pour dispositif. On parle alors de service composite, comme indiqué en **Figure 9**. Un service composite exporte deux interfaces : une interface dite fonctionnelle qui permet de communiquer avec les fonctionnalités proposées par le service, une interface dite structurelle permet d'accéder aux informations relatives à un assemblage et de le modifier. Les composants sondes et puits permettent d'intercepter et de déclencher des événements et des appels de méthodes dans l'assemblage géré par le container via ces interfaces [hourdin08].

5.2.2 Architecture globale de CONTINUUM

Notre approche des systèmes ubiquitaires repose sur trois niveaux (cf **Figure 10**) : (1) l'environnement accessible à travers l'infrastructure telle que nous venons de la présenter, (2) l'application réalisée comme une composition des services composant cette infrastructure et (3) le mécanisme de prise en compte du contexte. Nous avons vu dans la section précédente que l'utilisation des services pour dispositifs et leurs compositions dans le modèle SLCA permettent de construire des applications cohérentes avec leur infrastructure logicielle (c'est-à-dire qu'elles sont composées de services pour dispositifs présents dans l'infrastructure), avec entre autres la prise en compte de l'apparition/disparition de dispositifs. Nous présentons maintenant le troisième niveau de l'architecture de CONTINUUM : la prise en compte du contexte qui repose sur les niveaux (1) et (2)

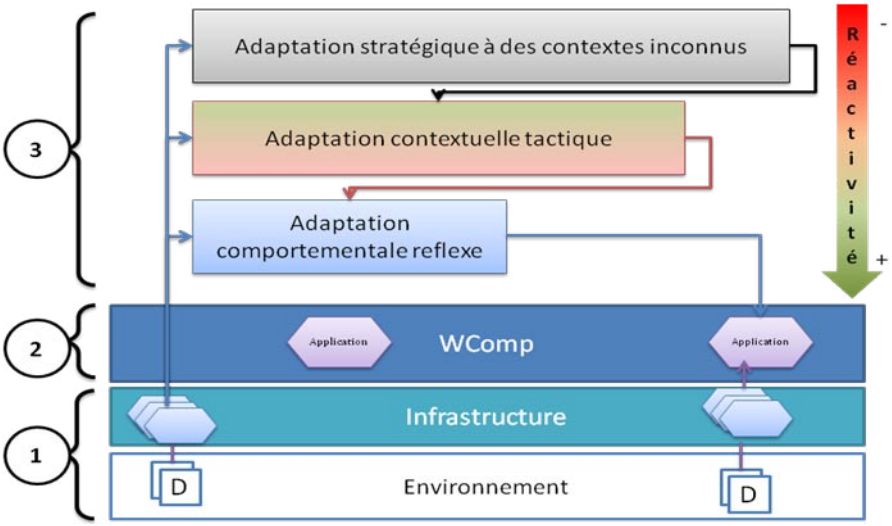


Figure 10. L'architecture de la plate-forme CONTINUUM reprenant le modèle des comportements cognitifs de Rasmussen.

Rasmussen, spécialiste reconnu en psychologie cognitive, décompose l'activité humaine en trois niveaux de comportements cognitifs [Rasmussen86]. Le premier niveau correspond aux comportements basés sur les habiletés : les réflexes. Ces comportements caractérisent des actions effectuées de manière inconsciente en réaction à des phénomènes perçus. Le second type de comportement s'appuie sur des ensembles de règles apprises avec l'expérience : les procédés souvent appelés connaissance procédurale. A ce niveau, il s'agit pour l'humain de choisir le bon ensemble de règles parmi les candidats correspondant à une situation connue. Le troisième type de comportement repose sur les connaissances profondes : les savoirs. Il s'agit d'élaborer un plan d'actions pour répondre à une situation inconnue.

Par analogie avec la décomposition de Rasmussen, nous proposons de structurer la gestion du contexte sous forme d'une architecture comportementale hiérarchique à trois niveaux. Le niveau N repose sur les mécanismes offerts par le niveau N-1 et peut agir sur la couche juste en dessous.

Chaque niveau poursuit sa propre dynamique avec ses abstractions, et par conséquent son niveau de complexité en rapport avec le nombre de couches intermédiaires entre le dit niveau et l'infrastructure. La réactivité est d'autant plus faible que cette complexité est élevée.

Ainsi, le premier niveau est appelé niveau réflexe. Il offre une dynamique au plus proche de celle de l'environnement. Ce niveau assure un temps de réponse fixé et réagit à des informations potentiellement partielles sur son environnement. Ces informations ne sont pas mémorisées. En ce sens, il offre de forte similitude avec les comportements reposant sur les habilités de Rasmussen.

Le second niveau, le niveau tactique, est capable d'identifier la situation courante parmi un ensemble de situations connues spécifiées à la conception, puis de sélectionner le plan d'actions correspondant. Son nombre d'états étant fini, son temps d'exécution est borné. Il permet de mettre en place en fonction de la situation de nouveaux réflexes au niveau inférieur. Il correspond au comportement basé sur les procédés de Rasmussen.

Le troisième niveau, le niveau stratégique, propose des mécanismes permettant d'adapter l'application à des situations inconnues. Il repose sur une vue globale du système et de son environnement. Ce niveau n'offre pas un temps de traitement borné. Il peut faire intervenir l'utilisateur dans la résolution d'un problème inconnu dans l'optique de déployer dans le niveau tactique de quoi identifier cette situation ou bien procéder par apprentissage automatique. Ce niveau est assimilable au comportement basé sur les savoirs de la décomposition de Rasmussen.

Ayant présenté l'approche hybride de l'architecture CONTINUUM, le modèle SCLA sur lequel repose notre infrastructure, ainsi que les différents niveaux de notre mécanisme de prise en compte du contexte, nous allons décrire en détail la projection de ces différents modèles sur le modèle d'implémentation du projet CONTINUUM.

6 Détail de l'architecture CONTINUUM : Modèle d'implémentation

Le modèle d'architecture présenté précédemment permet de comprendre l'orientation globale de l'architecture qui sera mise en œuvre dans le projet CONTINUUM. Regardons maintenant en détail le modèle d'implémentation qui permettra de répondre à la problématique de la continuité de services, finalité de ce projet. Pour rappel, les objectifs de l'architecture sont de fournir une solution orientée intergiciel pour :

- permettre l'hétérogénéité technologique et sémantique des nombreux dispositifs de l'environnement,
- gérer la variabilité du système ambiant et donc de prendre en considération les dynamiques des différents constituants du système,
- permettre à l'utilisateur de garder le contrôle sur les décisions du système.

Dans la suite, nous nous plaçons uniquement dans le monde logiciel. Nous faisons donc abstraction de la couche environnement et de l'infrastructure matérielle. Ce choix facilite les explications sans pour autant nous éloigner de la réalité : un système informatique ne peut percevoir son environnement qu'au travers des dispositifs de l'infrastructure et les données de ces dispositifs ne sont manipulables qu'au niveau logiciel de l'infrastructure.

6.1 Le socle : l'intergiciel WComp

Comme nous l'avons présenté précédemment, une application est représentée par un container WComp et constituée à la fois de composants et de composants proxy. Ces derniers sont les représentants locaux, dans le container de l'application, des services pour dispositifs répartis dans l'infrastructure.

Les services pour dispositifs reposent sur le standard UPnP qui garantit l'interopérabilité technique entre ces différents services. Le choix d'UPnP, face au deuxième standard de web services pour dispositifs qu'est DPWS, se justifie par une plus grande maturité d'UPnP. Cependant, l'intégration de composant proxy pour des services DPWS dans WComp n'est pas exclue. Il est de plus possible de faire cohabiter les deux types de proxy dans un même container et donc dans une même application.

UPnP Wizard Designer est une des briques essentielles de l'intergiciel WComp. En effet, même s'il est possible de générer via un IDE les composants proxy des différents services UPnP de l'infrastructure, cette génération nécessite une intervention humaine. Or, la grande variabilité de l'infrastructure de notre système ambiant ne nous permet pas de générer et d'instancier les composants proxy une fois pour toute. Il est nécessaire que cette génération se fasse au runtime de manière à garantir la dynamique des différents services UPnP. C'est le rôle de l'UPnP Wizard Designer présenté à la **Figure 11**. Celui-ci, à l'aide des mécanismes de gestion d'apparition et de disparition d'UPnP, est averti des changements de l'infrastructure. Il peut donc répercuter ces changements au niveau du container de l'application. Chaque fois qu'un nouveau service UPnP apparaît, l'UPnP Wizard instancie un nouveau

composant proxy lié à ce service dans le container de l'application. Et de manière similaire, quand un service UPnP disparaît de l'infrastructure, il supprime le composant proxy du container.

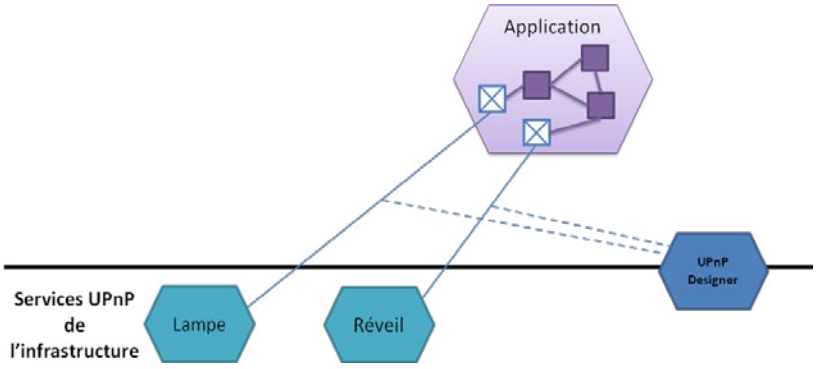


Figure 11. Socle de l'architecture CONTINUUM : un monde de services.

La dynamique d'apparition et de disparition de chaque service UPnP est alors répercutée au sein de l'application qui voit apparaître et disparaître les proxys correspondants.

6.2 Niveau réflexe : Les aspects d'assemblages

Le niveau réflexe doit offrir une dynamique au plus proche de celle de l'environnement (représenté dans l'application par l'apparition et la disparition des composants proxy) et par conséquent une forte réactivité avec des temps de réponses faibles. Pour ce faire, ce niveau repose lui-même sur un cas particulier de décomposition comportementale dans laquelle tous les comportements sont au même niveau et sans hiérarchie. Pour réaliser ces comportements, nous proposons une approche originale : les Aspects d'Assemblages. Ces Aspects d'Assemblages (AA) permettent de reconfigurer des applications respectant le modèle SLCA sans connaissance a priori des services découverts et utilisés à l'exécution de manière réactive. Comme leur nom l'indique, les AA sont tirés de la programmation par aspects.

La programmation par aspects (AOP) a été proposée par Kiczales [Kiczales97]. Il s'agit d'un paradigme qui permet de définir des préoccupations (extra)fonctionnelles transversales imbriquables (tissables) dans une application. La programmation par aspects est motivée par le constat suivant : malgré tous les efforts de modularité, il existe toujours un fort entrelacement entre le code métier d'une application et celui des préoccupations transverses (sécurité, monitoring …). L'idée de la programmation par aspects est donc de représenter de manière séparée, dans des aspects, ces préoccupations transverses. Il s'agit alors d'injecter du code dans du code. Cette injection se fait par l'intermédiaire d'un tisseur d'aspects. Un aspect se découpe en *points de coupe* et *greffons*, les premiers identifient « où » le code doit être tissé tandis que les seconds décrivent le code à injecter. La généricité de l'expression des points de coupes permet à un aspect d'être tissé en plusieurs endroits de l'application. La programmation par aspects permet donc de minimiser la dispersion du code en le regroupant dans des entités réutilisables. L'ensemble des attaches sur lesquelles peuvent être tissés des aspects s'appelle des points de jonctions. Selon les paradigmes sur lesquels repose la programmation par aspects, cet ensemble change de nature (objets, composants, code…) mais l'AOP offre toujours une bonne séparation des préoccupations, une bonne modularité transverse [charfi04].

L'AOP, en évoluant et en se couplant aux composants/services, a permis de se rapprocher des besoins de l'informatique ambiante, en apportant plus particulièrement la dynamicité du tissage. Toutefois, ce couplage peut être envisagé avec deux visions différentes. (1) Il peut s'agir de modifier un composant avec une approche invasive qui entraîne la perte des propriétés de boîte noire des composants. (2) Mais, l'AOP peut aussi être utilisée pour modifier la structure d'un assemblage. Navasa et al [Navasa02] mettent en avant les avantages suivants de cette approche : faciliter la conception de l'architecture de l'application, réduire les coûts de développement et de maintenance, réutiliser des sections d'architecture. Cependant, les approches existantes permettant de réaliser des reconfigurations structurelles d'assemblages de composants ne prennent pas en compte les évolutions de l'infrastructure sur laquelle repose l'application qu'elles adaptent. C'est à dire qu'elles ne permettent pas de composer dynamiquement les services qui apparaissent et qui disparaissent dans l'infrastructure logicielle du système.

Nous proposons donc un modèle tiré de la programmation par aspects appelé les Aspects d'assemblages (AA) [Cheung09] et illustré par la **Figure 12**. Si l'AOP permet de modifier un programme, les AA permettent de modifier la structure d'assemblages de composants. Il s'agit de tisser des schémas d'adaptation en suivant le principe de modifications structurelles. Le tissage d'AA agit sur la structure d'assemblages des composants. Il va donc à l'encontre du principe d'encapsulation. Toutefois, ils ne violent pas la propriété de boîte noire des composants sur étagère utilisés. Les points de jonctions des AA sont les ports des composants logiciels. Comme dans la programmation par aspects classiques, les AA permettent d'exprimer les notions de point de coupe et de greffon. Un point de coupe indique « où » les modifications doivent intervenir tandis que le greffon décrit les modifications à apporter dans un assemblage de composants pour en modifier le comportement.

Les points de coupes sont définis comme un ensemble de filtres portant sur les points de jonction, les ports de l'assemblage sur lesquels va s'appliquer l'AA. En fait, il s'agit de filtres sur les métadonnées, noms de ports, types … Ces filtres construisent des listes de paramètres, de points de jonctions, qui vérifient les listes de variables associées aux greffons. Ces listes permettent d'interfacer les aspects d'assemblages avec l'application. Elles peuvent ensuite s'organiser selon diverses combinaisons et contiennent les ports sur lesquels les greffons sont tissés. Pour chaque combinaison générée, le greffon est dupliqué et les variables sont syntaxiquement remplacées dans le greffon afin de correspondre aux points de jonction de l'assemblage de base. Un schéma d'adaptation peut ainsi être instancié en plusieurs endroits de l'assemblage. Les points de coupes permettent d'exprimer divers assemblages à haut niveau d'abstraction.

Les greffons ne sont pas des morceaux de code à tisser dans du code. Ils sont définis comme des ensembles de composants et de liens à tisser dans un assemblage de composants. Ils sont donc considérés comme des fabriques d'assemblages de composants. Ces assemblages sont construits sur les points de jonctions filtrés. Pour ce faire, les greffons se composent d'un ensemble de règles permettant d'écrire des assemblages de composants. Ces règles permettent d'instancier des composants ou de réaliser des connexions entre composants. Ainsi, les instances de greffon générées par des fabriques (des sous-assemblages) sont ensuite composées les unes aux autres. Pour cela, les assemblages générés peuvent être en boîte grise, c'est-à-dire qu'une partie de la sémantique de l'assemblage généré est connue. Ceci est possible grâce à l'utilisation de composants sur étagère de

sémantique connue. Grâce à ces composants, nous pouvons alors réaliser ce que l'on appellera un tissage interne ou encore une fusion d'instance de greffon et donc d'assemblages de composants.

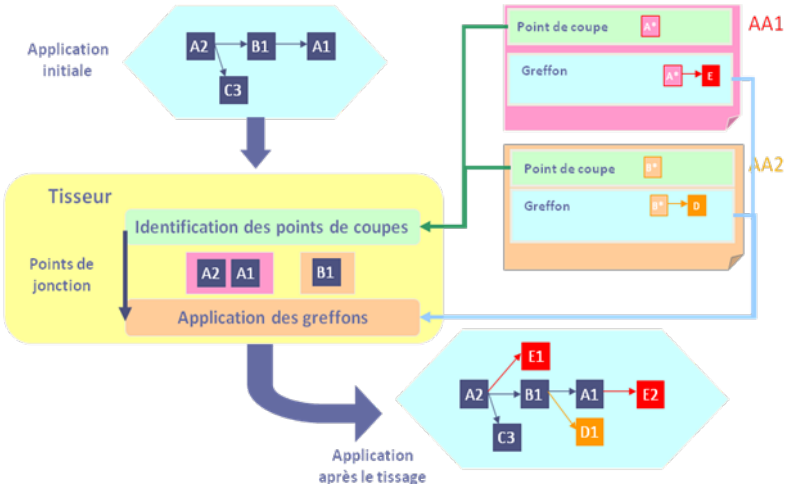


Figure 12. Détail du mécanisme de tissage des aspects d'assemblage.

A la manière des cycles d'automates qui consistent en des phases successives d'acquisition de données, d'exécution et finalement de productions de sorties, nous parlons de cycles de tissage. Durant un cycle, le tisseur d'AA prend en entrée un assemblage de composants, que l'on appellera assemblage initial, et un ensemble d'aspects d'assemblages. Il produit un assemblage final, un assemblage adapté. Un cycle de tissage peut être déclenché de deux manières :

1. par une modification de l'ensemble des AA déployés sur le tisseur. Si un AA est appliqué, son retrait entraîne un nouveau cycle de tissage. Si un nouvel AA est déployé et peut être appliqué cela entraîne également un nouveau cycle de tissage.
2. La seconde manière de déclencher un cycle de tissage est l'ajout/retrait de composant dans l'assemblage donné en entrée au tisseur. Quand un nouveau composant apparaît ou disparaît, par exemple lorsque de nouveaux dispositifs entrent dans l'infrastructure de l'application, un nouveau cycle de tissage est lancé et seuls les AA qui peuvent être appliqués sur ce nouvel assemblage sont tissés.

Comme le montre la **Figure 13**, un tisseur d'AA, appelé AA Designer, est associé à l'application de manière à l'adapter en fonction des changements de l'infrastructure et donc au rythme des apparitions et des disparitions des composants proxy.

Un point essentiel pour la réactivité de ce mécanisme est qu'il ne requiert pas d'informations relatives à l'infrastructure logicielle : avec une dynamique qui lui est propre, l'infrastructure impose son rythme via ce type de déclenchement. Nous offrons donc un mécanisme d'adaptation aux variations de l'infrastructure en respectant la dynamique de cette infrastructure. De plus, ce type de déclenchement d'adaptations permet de construire de manière opportune, à partir d'une infrastructure, une application selon une approche « bottom-up ».

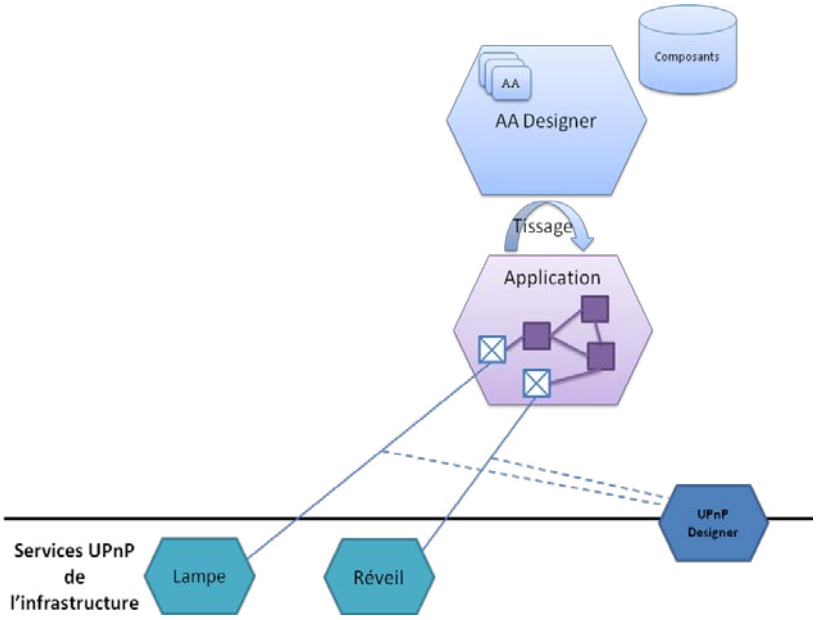


Figure 13. Niveau réflexe de l'architecture CONTINUUM.

6.3 Niveau tactique : Gestion du contexte et des connaissances

Le niveau tactique se trouve au deuxième niveau de l'infrastructure et s'appuie sur le niveau réflexe. Il n'a pas besoin d'une dynamique conforme à celle de l'application mais doit quand même être capable de réagir dans un temps raisonnable, correspondant aux différentes dynamiques qui régissent ce niveau.

Du point de vue de l'architecture, le niveau tactique est représenté par deux éléments distincts : le gestionnaire du contexte et la base de connaissances (**Figure 14**).

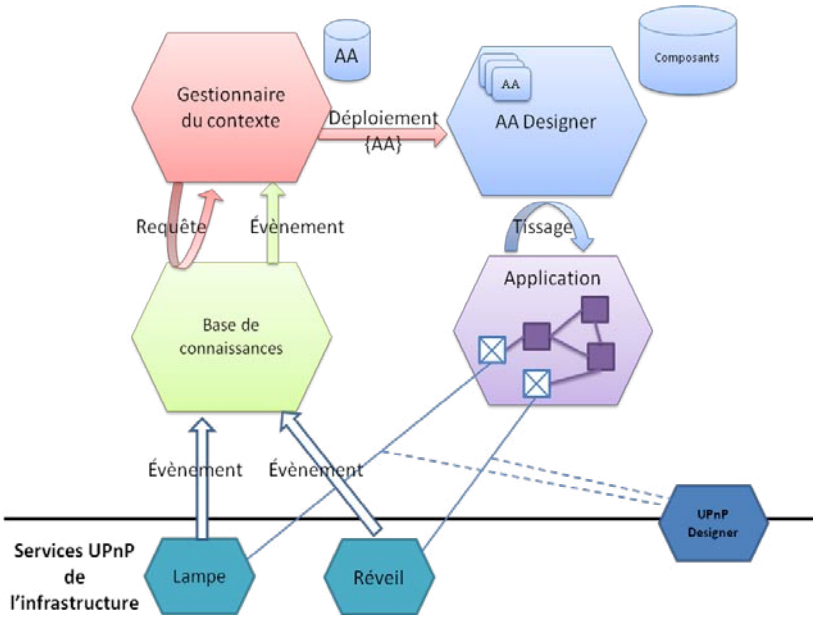


Figure 14. Niveau tactique de l'architecture CONTINUUM.

6.3.1 Le gestionnaire du contexte

Pour comprendre les tâches du gestionnaire du contexte (GC), il faut revenir à la définition du contexte (cf. deuxième section de ce livrable). Nous avons défini que durant la phase de modélisation

et de conception de l'application, le designer devait modéliser les différentes préoccupations pouvant impacter le bon fonctionnement de l'application sous forme de contexte, c'est-à-dire (1) un ensemble d'entités d'intérêts et (2) un ensemble de prédicats portant sur ces entités et permettant d'identifier dans quelle situation du contexte l'application se trouve.

Le rôle du GC est alors simplement d'identifier la situation courante pour l'ensemble des préoccupations, donc des contextes. Il y a donc autant de situations courantes que de contextes. Pour ce faire, le GC interroge la base de connaissances de manière à vérifier les différents prédicats issus de la modélisation des différents contextes. Une fois qu'une situation d'un contexte donné est identifiée, le GC déploie sur l'AA Designer les AA attachés à cette situation après avoir pris soin de retirer du Designer les AA correspondant à la situation précédente de ce même contexte.

En résumé, le GC ne fait que mettre à jour les AA de l'AA Designer. Il n'intervient pas directement sur l'application. Cette indirection permet d'avoir une adaptation au contexte dont le rythme est indépendant du rythme de l'infrastructure. Les changements de situations provoquant alors une adaptation du mécanisme d'adaptation, c'est-à-dire de l'ensemble des AA du Designer, et non de l'application elle-même. Cette dernière continue à s'adapter en fonction du mécanisme déployé dans le niveau réflexe de l'architecture CONTINUUM.

Enfin, il est important de noter que le GC n'a pas qu'une seule dynamique mais une multitude de dynamiques correspondant aux dynamiques des différents contextes surveillés.

6.3.2 La base de connaissances

La base de connaissances (BdC) ne sera pas détaillée dans ce livrable car le travail sur les aspects sémantiques et modélisation des connaissances font l'objet de la Tâche 3 du projet. Cependant un résumé décrivant cette approche BdC est fournit en annexe.

La BdC est le deuxième élément du niveau tactique. Son objectif est de maintenir un modèle de l'environnement facilement manipulable, à l'aide d'un langage décalratif, de manière à répondre aux requêtes du GC et ainsi lui permettre d'identifier les différentes situations des différents contextes. Pour ce faire, la BdC est à l'écoute de l'environnement c'est-à-dire que la BdC est averti de l'ensemble des évènements issus de l'infrastructure logicielle comme, par exemple, l'apparition ou la disparition de services UPnP.

Un élément important est que c'est la BdC qui pilote indirectement les dynamiques du GC. En effet, le GC n'évalue pas en permanence l'ensemble des prédicats des contextes qu'il gère. L'évaluation des prédicats est en fait conditionnée à la réception d'un ou plusieurs évènements en provenance de la BdC. Celle-ci peut émettre deux grandes catégories d'évènements :

1. Les évènements en provenance de l'infrastructure logicielle. Ici la BdC ne fait que transférer les évènements qu'elle reçoit au GC. Bien entendu, la BdC ne transfère pas tous les évènements mais seulement ceux susceptibles d'intéresser le GC. Elle joue donc ici un rôle de filtre entre l'infrastructure logicielle et le GC.
2. Les évènements issus de son propre traitement sur ses connaissances. En effet, on peut imaginer que différents traitements soient déportés sur la BdC et que celle-ci informe le GC en fonction des résultats de ceux-ci.

Avec le transfert des évènements du premier type, la BdC ne fait que transmettre la dynamique de l'environnement au GC. Avec le deuxième type, ainsi que lorsqu'elle filtre un évènement en provenance de l'infrastructure logicielle, la BdC influence la dynamique du GC. Il sera alors important de vérifier que les temps de réponse de la BdC aux requêtes du GC ainsi que lors du transfert d'évènements soient compatibles avec les différentes dynamiques du GC.

6.4 Niveau stratégique : Intervention de l'utilisateur et apprentissage

Le niveau stratégique est le troisième niveau de l'infrastructure et s'appuie sur le niveau tactique. S'éloignant encore un peu dans les couches de l'infrastructure, il voit sa dynamique suivre la même orientation et être complètement décorrélée de la dynamique de l'application. Mais cela n'a pas d'impact sur son fonctionnement car le but de ce niveau est de réagir aux contextes et situations inconnus de manière à mettre à jour le niveau tactique et plus particulièrement le GC comme l'illustre la **Erreur ! Source du renvoi introuvable.**.

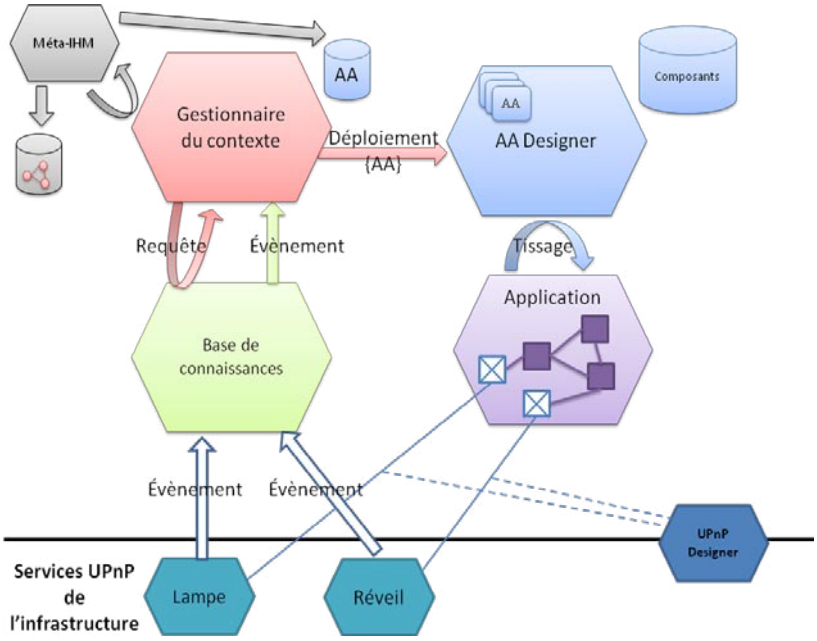


Figure 15. Niveau stratégique de l'architecture CONTINUUM.

Deux approches ont été identifiées pour réagir aux contextes et situations inconnues. La première consisterait en un mécanisme d'apprentissage tandis que la seconde ferait intervenir directement l'utilisateur. Dans le projet CONTINUUM, la première approche ne sera pas explorée car aucune équipe n'est spécialisée dans ce type d'approche. L'effort du projet sera donc déporté au niveau de la seconde approche qui fait l'objet de la tâche 4. Pour cette raison, et de manière similaire au cas de la BdC, nous ne détaillerons pas ici le fonctionnement de ce niveau mais nous donnerons seulement les grandes orientations pour comprendre l'approche et nous vous invitons à vous reporter aux livrables de la tâche 4 pour plus de détail.

L'approche choisie consiste donc à faire intervenir l'utilisateur pour identifier, modéliser et proposer des nouveaux contextes (avec les situations et les AA attachés à chaque situation). Cependant, si cela peut se faire avec les outils classiques pour un développeur, il en est tout autrement pour un utilisateur non expert (que nous appellerons utilisateur final). L'enjeu de ce niveau de l'infrastructure sera donc d'arriver à fournir à un utilisateur un moyen simple pour réaliser les tâches de

« reconfiguration du niveau tactique » de son système ambiant. Diverses étapes pourront être mises en place en fonction à la fois de la complexité de la tâche mais également en fonction de l'expérience de l'utilisateur. Par exemple, pour un utilisateur final débutant, la « reconfiguration » consistera simplement à modifier la liste des AA attachés aux situations alors qu'on pourrait imaginer aller jusqu'à la génération de nouveaux AA pour un utilisateur final expert.

6.5 Compléments transverses

Les compléments transverses correspondent à des éléments qui ont pour but d'enrichir les différents niveaux de l'architecture CONTINUUM mais qui ne respectent pas complètement le schéma de la **Figure 10**. Ces éléments passent outre l'organisation en couche qui impose qu'une couche de niveau n+1 ne peut agir que sur une couche de niveau n.

Les deux types de compléments transverses qui seront étudiés dans ce projet sont les Méta-IHM et les AA sémantiques.

6.5.1 Des Méta-IHM

Le but des Méta-IHM, qui font l'objet de la tâche 4, sont de permettre à l'utilisateur final de garder le contrôle sur l'ensemble du système ambiant. Pour cela, l'utilisateur doit au minimum pouvoir observer l'état de son système et, quand cela est pertinent, le modifier.

On pourrait penser que cette fonctionnalité est incluse dans le niveau stratégique que nous avons présenté dans la section précédente. Cela n'est que partiellement vrai. En effet, si le rôle des Méta-IHM correspond bien au rôle du niveau stratégique ainsi qu'à la dynamique de celui-ci, le but de ces IHM est bien de « contrôler l'ensemble du système » et nous insistons bien sur le mot « ensemble ». Par conséquent, si ces IHM dépendent du niveau stratégique elles ne pourraient qu'intervenir sur les niveaux stratégique et tactique mais pas sur le niveau réflexe.

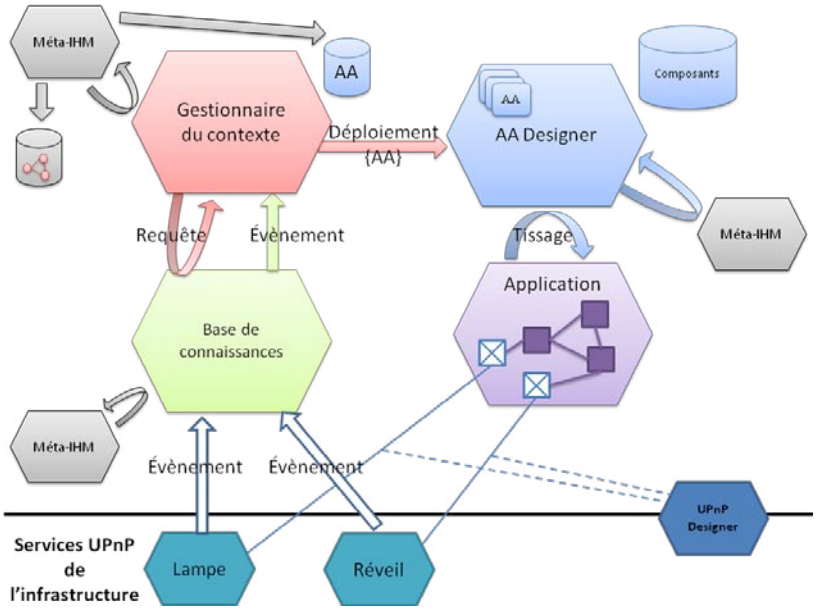


Figure 16. Les points de contrôle de l'architecture CONTINUUM.

Comme le montre la **Figure 16**, et sans entrer dans les détails (voir livrables de la tâche 4), nous illustrons dans les démonstrateurs de CONTINUUM trois types d'Méta-IHM.

- La première est celle du niveau stratégique qui permet l'observation mais également la modification du GC.
- La deuxième se situe au niveau de la base de connaissances pour permettre à l'utilisateur final d'observer les données de celle-ci et de pouvoir l'éditer (ajout, suppression et modification des données dans la base).
- La troisième s'interfacera avec l'AA designer dans l'optique de permettre à l'utilisateur de résoudre les interférences entre les différents AA telles qu'elles sont définies dans la sous-tâche 2.3

6.5.2 Des AA sémantiques

Le deuxième complément transverse que nous avons identifié intervient au niveau de la couche réflexe. Son objectif est d'enrichir le mécanisme d'identification des points de coupes qui aujourd'hui ne s'appuie que sur une analyse syntaxique des noms des composants. L'idée, au travers de ce complément, serait de proposer une extension du mécanisme d'identification pour lui permettre de faire une recherche sémantique à l'aide de la base de connaissances comme l'illustre la **Erreur ! Source du renvoi introuvable.**.

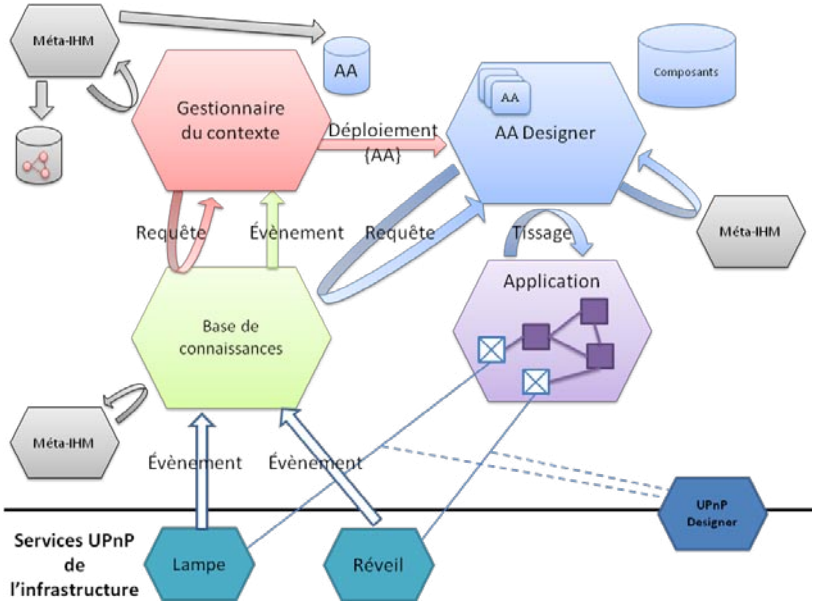


Figure 17. Extension sémantique des AA dans l'architecture CONTINUUM.

A l'aide de ce nouveau mécanisme, les AA verraient d'une part grandir leur généricité et d'autre part avoir des points de coupe contextualisés.

- Concernant la généricité, les AA pourraient en effet avoir des points de coupe indépendants de la syntaxe du nom de l'objet qu'ils recherchent (actuellement pour rechercher un écran les AA cherchent des objets ayant un nom de type « Display* »). Ils pourraient, par exemple, sur une recherche de « tous les écrans », obtenir de la BdC un objet qui se nomme « Tv » car celui-ci possède bien un écran.
- Concernant les points de coupe contextualisés, le but est de pouvoir formuler des requêtes du type « tous les écrans dans la même pièce que Bob », ceux à quoi la BdC est capable de répondre.

Le principal problème que pose ce complément transverse est qu'il lie l'AA Designer du niveau réflexe avec la BdC du niveau tactique. Cela aura pour effet potentiel de modifier la dynamique du niveau réflexe. Cependant, l'initiative des requêtes restant à l'AA Designer, nous pouvons déjà

affirmer que la BdC n'imposera pas sa dynamique au niveau réflexe. Il sera donc intéressant de vérifier, si oui ou non, ce lien particulier entre le niveau réflexe est le niveau tactique est pénalisant pour la réactivité de l'AA Designer et donc de l'adaptation de notre application.

7 Conclusion

Ce document est la fusion des livrables D2.1 et D2.2 des sous tâches 2.1 et 2.2 tels qu'ils avaient été définis dans le document de soumission. De ce fait, ce livrable a pour objectifs :

1. D'une part (Livrable D2.1), de définir un métamodèle du contexte et de montrer son application sur les scénarios industriel et prospectif définis dans la tâche 1.
2. D'autre part (Livrable D2.2), de définir les techniques permettant d'appliquer le plan de réaction aux variations du contexte et d'en préciser sa mise en œuvre pour le projet CONTINUUM.

Même si, comme nous allons le voir dans cette synthèse, nous retrouvons les contributions de chacun des livrables initiaux (D2.1 et D2.2) réparties respectivement en début et en fin de ce document, la fusion de ces deux livrables était nécessaire à la réalisation des travaux présentés ici.

Nous terminons donc ce livrable par une synthèse du travail effectué sur le début de la tâche 2 du projet et qui a été illustré tout au long de ce livrable.

Nous avons dans un premier temps défini la notion de système ambiant et les enjeux auxquels doivent répondre de tels systèmes pour garantir la continuité de service. Parmi les contraintes que nous avons identifiées, nous retenons les problématiques d'hétérogénéité tant technique que sémantique, la grande variabilité de l'infrastructure ainsi que les différentes dynamiques que les systèmes ambiants doivent prendre en considération.

Par la suite, après une rapide étude de la notion de contexte, nous avons présenté le modèle que nous avons défini et que nous utiliserons dans le cadre de ce projet. Nous avons vu au préalable que ce dernier repose sur les modèles des équipes académiques du projet en essayant de tirer parti de l'ensemble des avantages de chacune de leur approche.

Enfin, et cela conclura le travail initialement prévu dans le livrable D2.1, nous avons illustré l'utilisation de notre modèle du contexte sur les scénarios définis dans la tâche 1.

Dans la suite de ce livrable, nous nous sommes concentrés sur les aspects architecturaux et implémentationnels de notre approche. Nous nous sommes dans un premier temps intéressés aux intergiciels de prise en compte du contexte et à leur décomposition fonctionnelle. Puis, après avoir identifié plusieurs limitations de ces approches, nous avons étudié une autre approche : « la décomposition comportementale ».

En partant de cette approche de décomposition en comportement, nous avons montré comment devait évoluer les intergiciels de prise en compte du contexte pour garantir la maîtrise des dynamiques de cette prise en compte.

Puis, en nous appuyant sur les résultats précédents, nous avons défini une approche hybride permettant à la fois de garantir les dynamiques de prise en compte du contexte via une décomposition comportementale et à la fois de simplifier le développement d'applications contextuelles en respectant une décomposition fonctionnelle au sein de chaque comportement.

Par la suite, nous avons présenté les modèles LCA (de composition locale) et SLCA (une extension orienté service du modèle précédent) sur lesquels reposera l'implémentation de la plate-forme CONTINUUM.

Enfin, dans la dernière partie, nous avons détaillé l'ensemble de la plate-forme CONTINUUM. Celle-ci est constituée d'un ensemble de mécanismes organisés selon des niveaux reposant les uns sur les autres à la manière du modèle de [Rasmussen86]. Les quatre niveaux principaux de notre plate-forme sont :

1. Le socle qui réutilise l'intergiciel WComp.
2. Le niveau réflexe qui, à l'aide des aspects d'assemblages, permet d'adapter l'application avec une dynamique conforme à l'évolution de l'infrastructure.
3. Le niveau tactique qui s'appuyant sur une base de connaissances permet une gestion du contexte de plus haut niveau avec une dynamique différentes de celle de l'infrastructure.
4. Le niveau stratégique, avec encore une nouvelle dynamique, permet l'intervention de l'utilisateur à tout moment dans le cycle d'adaptation.

8 Bibliographie

[Acciarri05] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, R. Rosati. Quonto: Querying ontologies. In AAAI, 2005.

[Baader07] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider. The description Logic Handbook: Theory, Impolementation, and Applications. Cambridge University Press, 2003.

[Behlouli06] Chantal Taconet, Nabiha Belhanafi Behlouli. " CAMidO : un intergiciel pour la sensibilité au contexte ", Ubimob'06 Atelier d'étude du contexte (2006).

[Benyelloul10] A. Benyelloul, F. Jouanot, M.C. Rousset. Conquer: an RDFS-based model for context querying. UbiMob'10, 6 èmes Journées Francophones Mobilité et Ubiquité - 7 au 9 juin 2010, Lyon, France.

[Bottaro07] Andre Bottaro, Johann Bourcier, Clement Escoffier, Philippe Lalanda, "Context-Aware Service Composition in a Home Control Gateway ", 4th IEEE International Conference on Pervasive Services (ICPS'07), Istanbul, Turkey, Jully 2007.

[Brézillon02] Brézillon P. (2002) Expliciter le contexte dans les objets communicants. In: C. Kintzig, G. Poulain, G. Privat, P.-N. Favennec (Eds.): Les Objets Communicants. Hermes Science Editions, Lavoisier, Chapter 21, pp. 295-303.

[Brézillon05] M. Bazire et P. Brézillon. "Understanding Context Before Using It", Modeling and Using Context : 5th International and Interdisciplinary Conference CONTEXT 2005.

[Brooks91] R.A. Brooks. "Elephants don't play chess". Designing autonomous agents: Theory and practice from biology to engineering and back, 1991.

[Brown96] P.J. Brown, The Stick-e Document: a framework for creating Contextaware applications. Electronic Publishing '96 (1996) 259-272

[Brown97] P.J. Brown, J.D. Bovey, X. Chen. Context-Aware applications : From the Laboratory to the Marketplace. IEEE Personal Communications, 4(5) (1997) 58-64

[Bryzon01] Bryzon, "Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents". MIT.PhD Thesis 2001.

[Bussière07] N. Bussière, D. Cheung-Foo-Wo, V. Hourdin, S. Lavirotte, M. Riveill, and J.Y. Tigli. Optimized contextual discovery of web services for devices. In IEEE Int. Workshop on Context Modeling and Management for Smart Environments, volume 2, pages 707–712, October 2007.

[Calvanese07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati. Tractable reasoning and efficient query answering in description logics: the DL-lite family. Journal of Automated reasoning, 39(3):385-429, 2007.

[Cardoso09] R. S. Cardoso. "Intergiciel orienté services pour la protection de la vie privée dans les systèmes d'informatique diffuse" , PhD thesis, Université PARIS-VI, Jun 2009.

[Charfi04] A. Charfi, M. Mezini. "Aspect-Oriented Web Service Composition with AO4BPEL". Web Services: European Conference, ECOWS 2004, Erfurt, Germany, September 2004 .

[Chen00] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[Chen03] G. Chen, D. Kotz. “Context-sensitive resource discovery”, In Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, pages 243–252, Fort Worth, Texas, March 2003.

[Cheung09] Daniel Cheung-Foo-Wo. “Adaptation Dynamique par Tissage d’Aspects d’Assemblage ”, PhD thesis, Université de Nice - Sophia Antipolis, March 2009.

[Coutaz05] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. “Context is key”, ACM, 48(3) :49–53, 2005.

[Crowley02] J. L. Crowley, J. Coutaz, G. Rey, P. Reignier. Perceptual Components for Context Aware Computing. In Proc. Ubicomp (Ubiquitous Computing) 2002, pp 117-134.

[David06] P.C. David, T. Ledoux. “Une approche par aspects pour le développement de composants Fractal adaptatifs”, RSTI-Série L'Objet (RSTI-Objet) 12.2-3 (2006).

[Dey99] Anind K. Dey, Daniel Salber, Masayasu Futakawa and Gregory D. Abowd.An Architecture To Support Context-Aware Applications GVU Technical Report GIT-GVU-99-23. June 1999.

[Dourish01] P. Dourish. What we talk about when we talk about context. Submitted to HCI Journal for publication in special issue on context-aware computing, 2001.18p

[Fact] Fact++, http://owl.cs.manchester.ac.uk/fact++/.

[Ferry08] Nicolas Ferry, Stéphane Lavirotte, Gaëtan Rey, Jean-Yves Tigli. "Adaptation Dynamique d'Applications au Contexte en Informatique Ambiante" Research Report I3S (Université de Nice - Sophia Antipolis / CNRS), number I3S/RR-2008-20-FR, 36 pages, Sophia Antipolis, France, oct 2008

[Ferry09] Nicolas Ferry, Vincent Hourdin, Stéphane Lavirotte, Gaëtan Rey, Jean-Yves Tigli, and Michel Riveill. “Models at Runtime : Service for Device Composition and Adaptation”, In 4th International Workshop Models@Run.Time at Models 2009 (MRT’09), October 2009.

[Floch06] J. Floch et al. “Using architecture models for runtime adaptability”, IEEE software 23.2 (2006).

[GrandDictionnaire] Office québécois de la langue française. http://www.granddictionnaire.com.

[Gu05] T. Gu, H. Pung, and D. Zhang. “A service-oriented middleware for building context-aware services”, Journal of Network and Computer Applications, 2005.

[Hong02]
Hong, J. I. 2002. The context fabric: an infrastructure for context-aware computing. In CHI '02 Extended Abstracts on Human Factors in Computing Systems (Minneapolis, Minnesota, USA, April 20 - 25, 2002). CHI '02. ACM, New York, NY, 554-555. DOI= http://doi.acm.org/10.1145/506443.506478

[Hourdin08] Vincent Hourdin, Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Michel Riveill. "SLCA, composite services for ubiquitous computing" in Proceedings of the 5th International Conference on Mobile Technology, Applications and Systems (Mobility), pages 8, Taiwan, 10-12 september 2008

[Issarny07] Issarny, V., Caporuscio, M., and Georgantas, N. “A Perspective on the Future of Middleware-based Software Engineering”, In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 244-258.

[Issarny08] D. Athanasopoulos, A. Zarras, V. Issarny, E. Pitoura, and P. Vassiliadis. “CoWSAMI: Interface-aware context gathering in ambient intelligence environments”, Pervasive and Mobile Computing, 2008.

[Jang05] S. Jang, W. WOO. Unfified context representing user-centric context. ubiComp workshop E90-D, pages 26-34, 2005.

[Jena] Jena – a semantic web framework for java. http://jena.sourceforge.net.

[Jung07] Jung, J.Y. and Park, J. and Han, S.K. and Lee, K., “An ECA-based framework for decentralized coordination of ubiquitous web services”, Information and Software Technology, 2007.

[Kiczales97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. “Aspect-oriented programming”, ECOOP, SpringerVerlag, 1997.

[Lavirotte05] S. Lavirotte, D. Lingrand, and J.Y. Tigli. Définition du contexte : fonctions de coût et méthodes de sélection, In Proceedings of the 2nd French-speaking conference on Mobility and ubiquity computing, pages 9–12. ACM, 2005.

[Lei02] H. Lei, D.M. Sow, J.S. Davis, G. Banavar, and M.R. Ebling. “The design and applications of a context service”. ACM SIGMOBILE Mobile Computing and Communications Review, 2002.

[MacKenzie93] I.S. MacKenzie and C. Ware. “Lag as a determinant of human performance in interactive systems”, In Proceedings of the INTERACT’93 and CHI’93 conference on Human factors in computing systems, pages 488–493. ACM, 1993.

[McKinley04] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, “A Taxonomy of Compositional Adaptation”, technical report numéro MSU-CSE-04-17, juillet, 2004.

[Navasa02] A. Navasa et al. “Aspect oriented software architecture: a structural perspective”, 2002.

[Nigay03] Laurence Nigay, Emmanuel Dubois, P. Renevier, Laurence Pasqualetti, J. Troccaz. Mixed Systems: Combining Physical and Digital Worlds. Dans : HCI International, Crete, Greece, 22/06/03-27/06/03, Lawrence Erlbaum Associates, p. 4, juin 2003.

[OWL] W3C. OWL 2 web ontology language profiles. http://www.w3.org/2004/OWL, 2009.

[Pauty04] Julien Pauty, Paul Couderc, and Michel Banâtre. Synthèse des méthodes de programmation en informatique contextuelle. Technical Report 1595, IRISA, Janvier 2004.

[Pascoe98] J. Pascoe, Adding Generic Contextual Capabilities to Wearable Computers. 2nd International Symposium on Wearable Computers (1998) pages 92-99.

[Protégé] Protégé, http://protege.stanford.edu/.

[Racerpro] Racerpro. http://www.racer-systems.com

[Rasmussen86] J. Rasmussen.” Information processing and human-machine interaction: an approach to cognitive engineering”, 1986.

[Ryan97] N. Ryan, J. Pascoe, D. Morse, Enhanced Reality Fieldwork : the Context-Aware Archeological Assistant. V. Gaffney, M. van Leusen, S. Exxon Computer Applications in Archeology (1997).

[RDF] RDF, http://www.w3.org/RDF.

[RDFS] RDFS, http://www.w3.org/TR/2004/REC-rdf-schema-20040210.

[Rey05] G. Rey. “Contexte en Interaction Homme-Machine : le contexteur”, Communication Langagière et Interaction Personne-Système - Fédération IMAG - Université Joseph Fourier -

Grenoble I. Thèse préparée au sein de l'Université Joseph Fourier de Grenoble, 1er août, 2005, 186 pages.

[Roman02] M. Román et al. “Gaia: a middleware platform for active spaces”, ACM SIGMOBILE Mobile Computing and Communications Review 6.4 (2002), p. 65-67.

[Sacchetti05] D. Sacchetti et al. “The Amigo Interoperable Middleware for the Networked Home environment”,6th International Middleware Conference, Workshops Proceedings, Grenoble, France (2005).

[Saty96] M. Satyanarayanan. “Fundamental challenges in mobile computing”. Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing. ACM New York, NY, USA. 1996, p. 1-7.

[Schilit94a] B.N. Schilit, M. Theimer, Disseminating Active Map Information to Mobile Hosts, IEEE Network, (1994) 22 – 32.

[Schilit94b] B. Schilit, N. Adams, and R. Want. “Context-aware computing applications”, In Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on, pages 85–90, 1994.

[Singh05] M.P. Singh and M.N. Huhns. “Service-oriented computing : semantics, processes, agents”. John Wiley & Sons Inc, 2005.

[Sirin07] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyampur, Y. Katz. Pellet: A practical OWL-DL reasoner. Jouranl of Web Semantics, 5(2):51-33, 2007.

[Sirin07] E. Sirin, B. Parsia, B. Cuenca, A. kalyanpur, Y. Katz. Pellet: A practical OWL-DL reasoner. Journal of Web Semantics, 5(25):51-53, 2007.

[SPARQL] SPARQL, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[Strimpakou05] M. Strimpakou, I. Roussaki, C. Pils, M. Angermann, P. Robertson, and M. Anagnostou. “Context modeling and management in ambient-aware pervasive environments”, International Workshop on Location and Context-Awareness (LoCA 2005), Munich, Germany. Springer, 2005.

[Thevenin99] D. Thevenin, J. Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In Proceedings of INTERACT'99, 1999, pp. 110-117.

[Tigli09] J-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, M. Riveill. "Lightweight Service Oriented Architecture for Pervasive Computing", International Journal of Computer Science Issues (IJCSI), vol. 4, pages 1-9, 2009

[View02] T. View, “Reconfigurable context-sensitive middleware for pervasive computing,” Pervasive Computing, IEEE, vol. 1, no. 3, pp. 33-40, 2002.

[Ward97] Ward, A. Jones, A. Hopper, A New Location Technique for the Active Office. IEEE personnal Communications (1997) 42-47

[Weiser93] M. Weiser, Some Computer Science Problems in Ubiquitous Computing, Communications of the ACM, July 1993. (Reprinted as "Ubiquitous Computing". Nikkei Electronics; December 6, 1993; pp. 137-143.)

[Winograd 01] Terry winograd, Architectures for context. In Human-Computer Interaction ,Vol. 16, (2001). pp 402 - 419.

[Zhang04] C. Zhang et H.A. Jacobsen. “Resolving feature convolution in middleware systems”. In ACM SIGPLAN Notices 39.10 (2004).

[Hong02] J. Hong. “The context fabric: an infrastructure for context-aware computing”. In CHI, 2002.

9 Annexe : Représentations et modèles du contexte

Cette annexe présente un rapide état des lieux des travaux des différentes équipes académiques du projet sur les notions de représentations et de modélisations du contexte.

9.1 Modèle du contexte et des situations

Rey s'appuie sur la notion d'observable (une donnée captée par le système ou calculée à partir de données captées) qui servent à identifier les différentes entités du monde (utilisateur, …) ainsi que les rôles joués par ces entités ou les relations (relations spatiales, temporelles, ou autres) quelles entretiennent entre elles. De plus, une entité peut jouer plusieurs rôles et un rôle peut être joué simultanément ou séquentiellement par plusieurs entités.

Les relations, les rôles et les entités constituent les ensembles de définition d'un réseau de contextes et c'est dans ce réseau de contextes que s'exécute l'application que l'on souhaite adapter.

9.1.1 Réseau de contextes

Un réseau de contextes ***Rc***, est défini sur trois ensembles (***Roles***, ***Relations***, ***Entites***):

- ***Roles*** est l'ensemble des rôles considérés par le concepteur du système interactif. $\boldsymbol{Roles = \{r_1, r_2, \dots, r_n\}}$ où $\boldsymbol{r_i}$ est un rôle,
- ***Relations*** correspond à l'ensemble des relations entre les entités considérées par le concepteur du système interactif. $\boldsymbol{Relations = \{rel_1, rel_2, \dots, rel_n\}}$ où $\boldsymbol{rel_i}$ est une relation,
- ***Entites*** désigne l'ensemble des entités considérées par le concepteur du système interactif. $\boldsymbol{Entites = \{e_1, e_2, \dots, e_n\}}$ où $\boldsymbol{e_i}$ est une entité.

9.1.2 Contexte

Chaque nœud du réseau ***Rc*** correspond à un contexte ***Ci***.
Un contexte ***Ci*** est défini par le couple $\boldsymbol{(R_i, Rel_i)}$ où :

- $\boldsymbol{R_i}$ est l'ensemble des rôles joués par les entités, avec $\boldsymbol{R_i \in Roles}$. De plus, quel que soit $\boldsymbol{r \in R_i}$, il existe une entité $\boldsymbol{e \in Entites}$ telle que ***e*** joue le rôle ***r***.
- $\boldsymbol{Rel_i}$ est l'ensemble des relations (entre entités), avec $\boldsymbol{Rel_i \in Relations}$. De plus, quelle que soit la relation ***rel*** (d'arité j) $\boldsymbol{\in Rel_i}$, il existe j entités $\boldsymbol{e_1}$ à $\boldsymbol{e_j \in Entites}$ telle que les entités $\boldsymbol{e_1}$ à $\boldsymbol{e_j}$ entretiennent la relation ***rel***.

9.1.3 Changement de Contexte

De ces définitions et propriétés, il y a changement de contexte lorsque l'une des conditions suivantes devient vraie :

- L'ensemble $\boldsymbol{R_i}$ des rôles remplis change : apparition de rôles et/ou disparition de rôles.
- L'ensemble $\boldsymbol{Rel_i}$ des relations entretenues change : apparition de nouvelles relations et/ou disparition de relations.

9.1.4 Contexte = Réseau de Situations

De manière à détailler les évolutions qui pourraient avoir lieu sans provoquer un changement de contexte, Rey introduit la notion de situation.

Un contexte ***C = (R,Rel)*** est alors décrit comme un réseau de situations tel que toutes les situations de ***C*** partagent les mêmes ensembles de rôles ***R*** et de relations ***Rel*** où ***R*** $\subseteq$ ***Roles*** et ***Rel*** $\subseteq$ ***Relations***.

La **Figure 18** illustre la décomposition d'un réseau de contexte ***Rc*** où chaque contexte ***C*** est à son tour un réseau de situations.

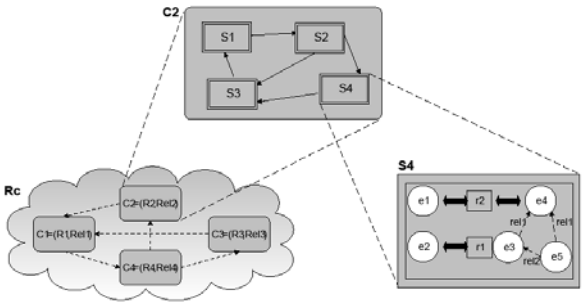


Figure 18. Décomposition d'un réseau de contextes en contextes et situations.

9.1.5 Situation

Au sein d'un contexte ***C = (R,Rel)***, une situation ***S*** est définie sur trois ensemble ***(Ent, AssoRoEnt, AssoRelEnt)***:

- ***Ent*** est l'ensemble des entités présentes dans ***S***, avec ***Ent*** $\subseteq$ ***Entites***,
- ***AssoRoEnt*** est l'ensemble des associations entre les rôles de ***R*** et les entités de ***Ent***. Rappelons que chaque rôle r_i est joué par au moins une entité et que chaque entité joue zéro, un ou plusieurs rôles,
- ***AssoRelEnt*** est l'ensemble des associations entre les relations de ***Rel*** et les entités de ***Ent***.

9.1.6 Changement de Situation

Au sein d'un contexte ***C = (R,Rel)***, il y a changement de situation (**Figure 19**) si l'une au moins des conditions suivantes est remplie :

- l'ensemble ***Ent*** des entités change : apparition et/ou disparition d'entité(s),
- l'ensemble ***AssoRoEnt*** des associations entre les rôles et les entités change : apparition et/ou disparition d'association(s) rôle-entité,
- l'ensemble ***AssoRelEnt*** des associations entre les relations et les entités change : apparition et/ou disparition d'association(s) relation-entité.

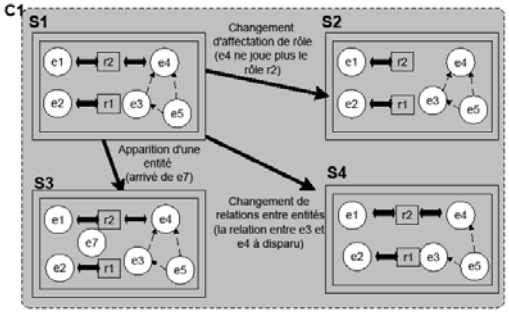


Figure 19. Changements de situation.

9.2 Les zones contextuelles

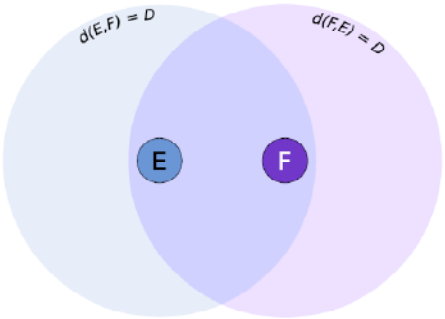
L'équipe Rainbow de l'I3S a choisi une approche locale permettant d'utiliser les informations contextuelles de chaque entité du système pour définir une relation de proximité contextuelle entre deux entités. Ainsi pour une entité de référence, son contexte sera composé de l'ensemble des entités considérées comme proche au regard des informations contextuelles de chacune.

Une zone contextuelle est donc définie par rapport à une entité. Elle est constituée de l'ensemble d'entités appartenant au contexte de cette entité. Elle correspond à des critères d'appartenance, ou de sélection, d'une entité au contexte d'une autre. Commençons par l'étude du cas de zones contextuelles symétriques pour lesquelles, lorsqu'une entité E est dans la zone contextuelle de l'entité F, il est supposé que F est aussi dans le contexte de E.

9.2.1 Zone contextuelle symétrique

Étudions un exemple simple basé sur un contexte géographique, toujours avec deux entités E et F. Une zone contextuelle Z est définie par une fonction de distance géométrique d et une distance maximale commune et constante D [Pauty04]. Elle est ainsi représentée par une zone géographique autour d'une entité, par exemple une sphère dans un repère orthonormé à trois dimensions. Le voisinage est ainsi défini par une distance géométrique simple. Si l'entité F est à une distance de E inférieure à D, elle fait partie de la zone contextuelle de E. Dans ce cas, E est à la même distance de F et fait aussi partie du contexte de F. Nous en tirons les équations suivantes, avec E et F représentant le contexte des entités E et F :

$$\begin{cases} d(E,F) \geq 0 \\ d(E,F) = 0 \Leftrightarrow E = F \\ d(E,F) = d(F,E) \\ d(E,F) \leq d(E,G) + d(G,F) \end{cases} \qquad \forall F,\ F \in Z(E) \Rightarrow d(E,F) \leq D \qquad (1.1)$$



$$d(E,F) = d(F,E) \leq D \qquad (1.2)$$

Figure 20. Illustration du contexte symétrique : zones géographique définies par une fonction de distance.

Comme on le voit dans la **Figure 20** ci-dessus, puisqu'une distance est une relation symétrique, quand E est dans le contexte de F, F est aussi dans le contexte de E. Les deux distances sont inférieures ou égales à la limite de la zone contextuelle D (Éq. 1.2). Lorsque des informations de localisation sont utilisées pour le contexte, une fonction de distance peut se révéler suffisante pour déterminer une zone contextuelle. Ca n'est pourtant par toujours le cas [Lavirotte05].

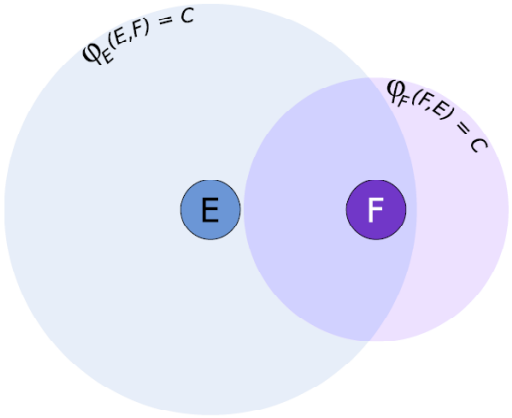
9.2.2 Zone contextuelle asymétrique

Considérons maintenant que la zone contextuelle est définie par une zone circulaire de diamètre propre à chaque entité. La **Figure 20** ci-dessous représente les zones contextuelles de E et F, avec un diamètre plus important pour E que pour F. Ainsi, F est visible par (dans le contexte de) E, mais E

n'est pas visible par (dans le contexte de) F. Le contexte ne se limite donc pas à une simple évaluation de distance, puisque la distance entre deux entités est une propriété symétrique et donc commutative (ÉQ. 1.1). Dans de nombreux cas, l'appartenance au contexte est asymétrique, et nécessite d'introduire les fonctions de coût j à la place des fonctions de distance. Elles sont définies par chaque entité, pour leurs propres besoins, et en relation avec leurs capacités. Par exemple, l'énergie nécessaire pour un voyage est une fonction de coût, puisque suivant l'altitude des points de départ et d'arrivée, l'énergie nécessaire pour aller de l'une à l'autre et inversement n'est pas identique ; pourtant la distance est par définition identique à l'aller et au retour.

Chaque entité définit sa zone contextuelle en utilisant sa fonction j, son contexte, et le contexte des autres entités. Dans les équations suivantes, nous notons j_E la fonction j de l'entité E. L'appartenance de F à la zone contextuelle (au contexte) Y de E pourra alors être calculée, non plus à partir d'une distance maximale D, mais d'un coût maximal C :

$$\begin{cases} \varphi_E(E,F) \geq 0 \\ \varphi_E(E,F) \neq 0 \Rightarrow E \neq F \\ \varphi_E(E,F) \leq \varphi_E(E,G) + \varphi_E(G,F) \end{cases} \qquad \forall F,\ F \in \Psi(E) \Rightarrow \varphi_E(E,F) \leq C \qquad (1.3)$$



$$\begin{cases} \varphi_E(E,F) \neq \varphi_F(F,E) \\ \varphi_E(E,F) \leq C \\ \varphi_F(F,E) > C \end{cases} \qquad (1.4)$$

Figure 21. Illustration du contexte asymétrique : zones géographiques variables définies par une fonction de coût.

Les propriétés des fonctions de coût (ÉQ. 1.3) diffèrent quelque peu par rapport à celles des fonctions de distance. La propriété de symétrie n'est bien sûr pas respectée, et on ne peut pas non plus déterminer que deux entités sont identiques si le coût est nul [Lavirotte05]. Comme on le voit dans la **Figure 21**, les zones contextuelles sont évaluées par chaque entité et ne sont pas identiques. L'entité E n'est pas dans le contexte de l'entité F parce que l'évaluation de leurs fonctions de coût ne donne pas les mêmes résultats (ÉQ. 1.4). La valeur de C importe alors peu, puisque les fonctions j des différentes entités ne fournissent pas les mêmes résultats. Le fait qu'elle soit constante et commune à toutes les zones contextuelles n'est plus nécessaire. On peut ainsi simplifier l'équation (ÉQ. 1.3) en faisant passer le C à l'intérieur des j, ce qui simplifie l'expression des zones contextuelles :

$$\forall F,\ F \in \Psi(E) \Rightarrow \varphi_E(E,F) \leq 0 \qquad (1.5)$$

Dans les cas où seule l'appartenance à la zone contextuelle est pertinente, une simplification des j peut encore être faite : un résultat booléen serait mieux adapté pour décrire une appartenance qu'une valeur numérique.

En déplaçant le test de comparaison (Éq. 1.5) à l'intérieur des j, on obtient une fonction valuée booléenne (Éq 1.6). Le parallèle entre la logique booléenne et la définition des zones contextuelles est pertinent : un élément (une entité) appartient (2) à un ensemble (une zone contextuelle), si la fonction de coût fournit un résultat positif.

$$\forall F,\ F \in \Psi(E) \Rightarrow \varphi_E(E,F) = 1 \quad (\text{ou } true) \tag{1.6}$$

Nous modélisons les zones contextuelles asymétriques et les fonctions j avec un métamodèle (**Figure 22**). Une zone contextuelle est définie par une entité en utilisant sa fonction j, son vecteur contextuel qui est constitué grâce aux observateurs de contexte, et les autres entités.

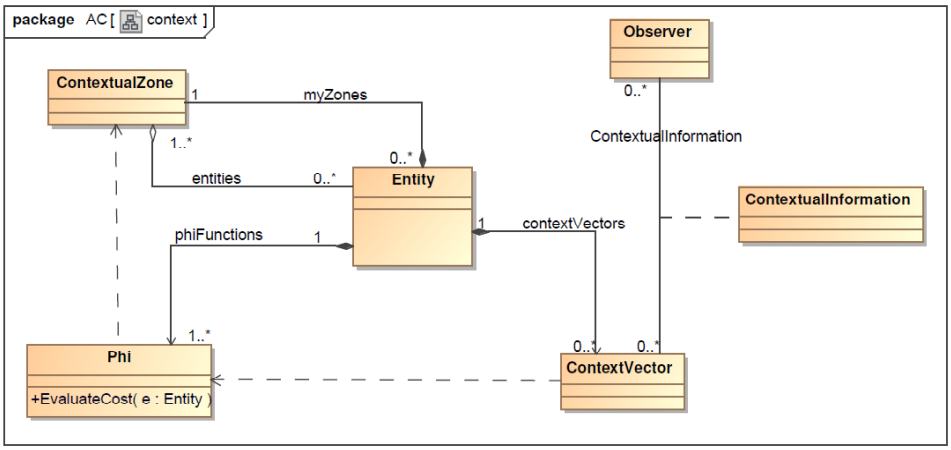


Figure 22. Meta-modèle des zones contextuelles asymétriques.

9.2.3 Sélection asymétrique du contexte

Puisque les zones contextuelles sont asymétriques, la sélection contextuelle, qui détermine les entités qui appartiennent à un contexte, peut se faire de deux façons par rapport à une entité E : on peut sélectionner les entités qui sont dans la zone contextuelle de E (sélection endo) ou les entités dont la zone contextuelle contient E (sélection exo) [Lavirotte05] **Figure 23**). On retrouve alors la notion de contexte relatif dans lequel le contexte est défini par rapport à une autre entité [Pauty04].

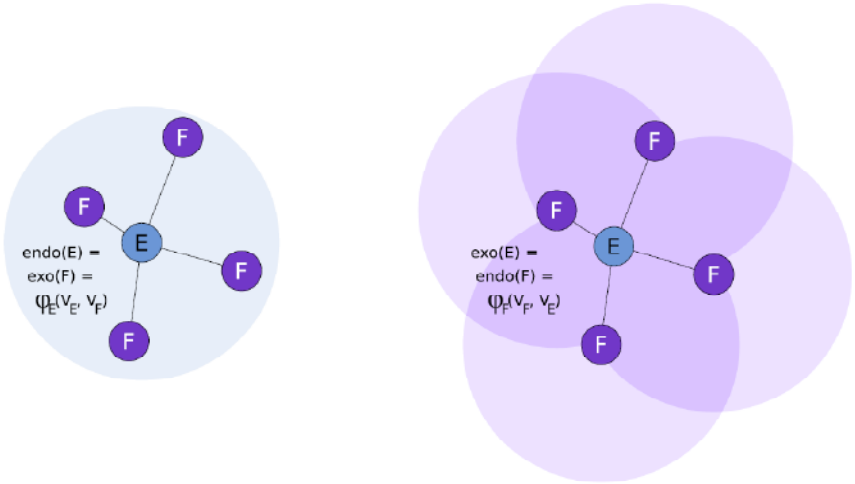


Figure 23. Endo et exo sélections du contexte de E.

La sélection endo du contexte, que nous appelons aussi l'endo contexte, noté endo(E) pour une entité E, permet d'obtenir les entités appartenant au contexte de E. Néanmoins, lorsque l'on se place du point de vue des autres entités (F), on peut dire que l'on fait partie de la zone contextuelle de E, ce qui est la définition de exo(F) (Cf. **Erreur ! Source du renvoi introuvable.**).

Tableau 1. Sélection asymétrique du contexte.

Point de vue de E		Point de vue de F	
$endo(E)$	F est dans mon contexte	Je suis dans le contexte de E	$exo(F)$
$exo(E)$	Je suis dans le contexte de F	E est dans mon contexte	$endo(F)$

Lorsque la sélection est effectuée à la fois en endo et en exo du point de vue de la même entité, elle est appelée sélection bilatérale (Éq. 1.7). La pertinence de l'utilisation d'un dispositif dans une application pourra par exemple être augmentée en utilisant une sélection bilatérale du contexte : si le dispositif est dans mon contexte, il peut m'être utile, et si je suis dans son contexte, je suis en mesure de l'utiliser.

Dans [Bussière07], nous avons mis en avant l'utilisation de l'exo contexte pour le contrôle d'accès dans la découverte des dispositifs. Être dans le contexte d'un dispositif signifie dans ce cas que l'on a le droit de découvrir et d'utiliser le dispositif. La sélection bilatérale fournit alors une liste de dispositifs présents dans l'infrastructure logicielle et pour lesquels une autorisation d'accès basée sur le contexte a été obtenue.

$$bilateral(E) = endo(E) \cap exo(E) \tag{1.7}$$

Nous avons rappelé dans cette section les principes de la notion de zone contextuelle de l'équipe Rainbow de l'I3S permettant d'utiliser les informations contextuelles de chaque entité du système pour définir une relation de proximité contextuelle entre plusieurs entités.

9.3 Conquer, interrogation et raisonnement sur le contexte

L'équipe Hadas du LIG propose un modèle du contexte qui s'appuie sur les standards du web sémantique. L'approche retenue représente un contexte d'exécution sous la forme d'une ontologie décrite dans le langage RDFS [RDFS] (Ressource Description Framework Schema). L'idée principale est de pouvoir disposer d'une approche déclarative pour modéliser et interroger les données du contexte. La manipulation comme l'interrogation du contexte peuvent ainsi exploiter les propriétés de raisonnement et d'inférences utilisées dans le contexte du web sémantique.

9.3.1 Le modèle CONQUER

Notre modèle du contexte est constitué de quatre classes de base dans le domaine de l'informatique ambiante, relatives aux concepts de Personne, Dispositif, Service et Tâche [Benyelloul10]. Chacune des classes est composée d'un ensemble de propriétés inspiré du modèle 5W1H [Jang05]. Ce noyau, constituant une métareprésentation du contexte peut être facilement enrichi par spécialisation des classes de base afin de capturer toute la sémantique nécessaire au contexte d'une application.

La description d'un contexte sous Conquer, est un ensemble de règles en RDFS, c'est-à-dire un ensemble de contraintes sur un vocabulaire exprimé en RDF et permettant d'exprimer la notion de classe, de propriétés, et les relations entre eux.

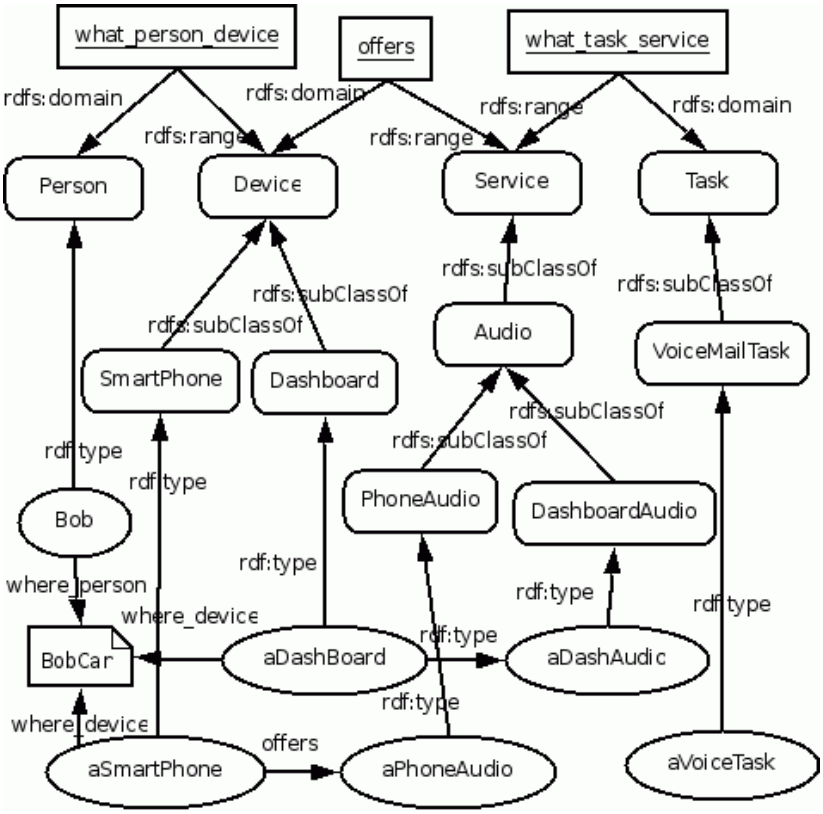


Figure 24. Exemple de modélisation d'un contexte.

9.3.2 Interroger le contexte

Conquer permet d'écrire les requêtes exprimables sous la forme d'une conjonction de prédicats. Par exemple, la requête "quels sont les dispositifs offrant un service de type audio dans la voiture de Bob ?" peut être définie comme suit en logique du premier ordre:

$$q(x): Device(x) \wedge where\_device(x,BobCar) \wedge offer(x,s) \wedge Audio(s)$$

L'évaluation de cette requête sur les données du contexte consiste à trouver les instances pour les différentes variables pour lesquelles l'expression en logique du premier ordre est substituée par les faits connus. Sur l'ensemble des faits exprimés dans la **Figure 24**, l'expression logique est satisfaite en considérant l'ontologie définie par le contexte et des axiomes d'inclusion de la forme :

$$SmarPhone \subseteq Device \quad DashBoard \subseteq Device$$
$$PhoneAudio \subseteq Audio \quad DashBoardAudio \subseteq Audio$$

De nouveaux faits sont alors inférés et l'évaluation de l'expression logique précédente donne le résultat {aSmarthPhone,aDashBoard}.

Les déclarations RDFS forment un ensemble de règles qui peuvent être appliquées en utilisant un algorithme de chaînage avant. Cependant, l'évaluation de requêtes peut aussi être réalisée à l'aide d'un langage de requêtes standard (de type SQL par exemple) sur un ensemble de faits. Conquer permet l'utilisation du langage d'interrogation déclaratif SPARQL [SPARQL] pour interroger les données du contexte. Ce langage est un standard du W3C pour interroger les données RDF (et donc RFDS). L'évaluation correcte des requêtes SPARQL est assurée par l'exécution du moteur SPARQL sur un ensemble de faits saturés. La requête logique précédente devient alors,

```
SELECT ?d WHERE { ?d type Device .
?d where_device BobCar .
?d offers ?s .
?s type Audio . }
```

Cette approche permet de poser, sous forme déclarative, des requêtes complexes sur les données du contexte. Les réponses à ces requêtes sont à la base de réactions adaptées au contexte.

9.3.3 Conquer WebService

Nous avons développé un web service : Conquer, qui implémente une base de faits RDFS et offre les opérations suivantes pour manipuler ces faits :

- *Add* : Ajout d'un triplet dans la base ;
- *Delete* : Suppression d'un triplet ;
- *Query* : Exécution d'une requête SPARQL sur la base.

Le service maintient une base de faits RDFS « saturée » ce qui permet de réutiliser des moteurs de requête SPARL existants, qui n'implémentent pas, pour la plupart, d'algorithmes de chaînage avant.

Nous avons aussi développé une interface web dans le but d'illustrer les fonctionnalités du service, en fournissant une représentation graphique des données, et des formulaires simplifiant les opérations d'édition et d'interrogation.

L'informatique ambiante implique des environnements très dynamiques et donc une nécessité de remodeler sans cesse le contexte. La représentation, l'exploitation et l'interrogation de connaissances demande des outils souples et réactifs. A la différence de canevas logiciel très complet comme Jena [Jena] ou Quonto [Acciarri05], tous deux capables de raisonner sur des masses de données, Conquer est un outil plus simple mais bien adapté aux données très dynamiques du contexte, aux dispositifs légers, et facilement extensible en fonction de l'application ciblée.