Adaptation au contexte par tissage d'aspects d'assemblage de composants déclenchés par des conditions contextuelles

Jean-Yves Tigli[*], Daniel Cheung-Foo-Wo[*,], Stéphane Lavirotte[*,***], Michel Riveill[*]**

** Laboratoire I3S, Equipe Rainbow*
CNRS / Université de Nice – Sophia Antipolis
930, route des Colles - BP 145 F-06903 Sophia Antipolis cedex
*** CSTB 290, route des Lucioles – BP 209 F-06904 Sophia Antipolis*
**** IUFM Célestin Freinet – Académie de Nice*
89, avenue George V F-06046 Nice cedex 1
{tigli, cheung, lavirott, riveill}@polytech.unice.fr

RÉSUMÉ. *Avec la multiplication des terminaux mobiles et des objets communicants dans notre vie quotidienne, nous observons depuis quelques années l'émergence d'applications faisant appel à la notion de contexte. Après un rapide rappel sur l'Informatique Mobile et Ambiante, nous présentons une modélisation de la problématique de l'interaction entre le système informatique, l'utilisateur et son environnement. Nous nous intéressons au contexte d'interaction et présentons des définitions de cette notion. Un modèle unifié et formel du contexte permet de décrire des fonctions de définition de la notion de voisinage. Ces dernières correspondent alors à la vérification de conditions sur le contexte nécessitant des modifications dynamiques de l'application. Après une revue des différentes approches de la littérature pour l'adaptation d'une application à son contexte, nous présentons une approche originale pour l'adaptation dynamique d'applications au contexte par sélection d'aspects d'assemblage et par composition dynamique de composants.*

MOTS-CLÉS : *adaptation, contexte, voisinage, proximité, aspect, composants.*

ABSTRACT. *Due to the multiplicity of mobile computer terminals and communicating objects in our daily life, we observe the emergence of applications using the notion of context. We propose a survey on mobile and ambient computing and then present a model for solving the problem of the interaction between the computer, the user and their environment. We propose a set of definitions of that notion. This is a unified and formal model that allows us to describe rigorously the notion of neighborhood. The latter corresponds to the validation of conditions on a context, which leads to modifications and adaptations of the application. Hence, after another survey on contextual software adaptation, we propose a software adaptation based on aspects of assemblies and dynamic composition of components.*

KEYWORDS: *adaptation, context, neighborhood, proximity, aspect, components.*

Revue. Volume 11 – n° 5/2006, pages 89 à 114

1.Introduction

Avec la multiplication des terminaux mobiles dans notre vie quotidienne, tels que PDA ou téléphone cellulaire, nous observons depuis quelques années l'émergence d'applications faisant appel à la notion de contexte. Nous abordons dans ce document les conséquences de la Mobilité sur le développement des applications logicielles et plus particulièrement sur les outils d'aide à la conception et à l'adaptation dynamique des applications en Informatique Mobile à leur contexte. Dans une première partie, nous établissons la définition par domaine de la notion de contexte pour l'Informatique Mobile et Ambiante. Après un rappel sur l'Informatique Mobile et Ambiante, nous présentons une modélisation de la problématique de l'interaction entre le système informatique, l'utilisateur et son environnement. Ceci nous conduit à expliciter 1) la notion de contexte en présentant différentes approches qui ont été développées jusqu'à maintenant et 2) notre extension de cette notion en termes de *zones contextuelles* et de fonctions de distance et de coût. La seconde partie vise à recenser les principes de l'adaptation logicielle mis en jeu pour la prise en compte de ces contextes, tels que, par exemple, l'introduction de composants spécifiques pour la capture, le traitement et l'extraction d'informations contextuelles. Nous présentons ensuite les mécanismes d'adaptation transversaux que nous avons mis en place pour répondre à cette problématique, mécanismes que nous illustrons finalement par quelques exemples.

2.Informatique Mobile et Ambiante

« *L'Informatique Mobile (Mobile Computing) repose sur l'étude de systèmes dans lesquels les composants informatiques peuvent changer d'emplacement et d'environnement dynamiquement* », selon (Roman *et al.*, 2000). Notons que cette définition n'est complète que si l'on précise les espaces dans lesquels les systèmes évoluent, en d'autres termes, l'environnement du système étudié. Par conséquent, (Roman *et al.*, 2000) distinguent plusieurs types d'environnement : continu, discret, logique ou physique. La thématique des premiers travaux sur l'Informatique Mobile reposant sur la mobilité du code fait donc référence à des entités logicielles évoluant dans un environnement *discret et logique*. C'est le cas d'un réseau de machines.

L'Informatique Mobile ne se réduit pas uniquement à cela. En effet, elle couvre également le spectre des contraintes induites par la mobilité des composants informatiques qui constituent le système. Ces « mutations » de l'infrastructure-support font apparaître de nouvelles contraintes souvent en opposition avec les solutions éprouvées et utilisées jusqu'ici dans de nombreuses disciplines de l'informatique. C'est ainsi que progressivement l'Informatique Mobile est devenue un thème abordé par de nombreuses communautés de chercheurs telles que celle des

IHMs, des plates-formes et des architectures logicielles, des ergonomes, des bases de données et des télécommunications et réseaux (Cépaduès, 2004). Quant à l'Informatique Ambiante (*Ubiquitous Computing*), elle correspond à l'évolution technique envisagée il y a une quinzaine d'années par (Weiser, 1991). Par opposition à l'informatique traditionnelle, la nouveauté tient à la capacité de ces systèmes à s'adapter en prenant en compte leur mobilité dans le milieu physique, la dynamique des interconnexions, le fait d'avoir un certain degré d'intelligence et le contexte d'exécution des applications (Lyytinen *et al.*, 2002). Mais avant toute chose, il convient de modéliser la notion de système Informatique Mobile et Ambiant.

2.1.*Modélisation des systèmes Informatiques Mobiles et Ambiants*

Si nous nous concentrons sur l'interaction entre le système informatique et son environnement, nous devons clairement distinguer les organes de traitement de l'information des organes d'interaction tels que les dispositifs d'entrée-sortie. Nous décrivons donc un système informatique mobile à partir de deux types d'entités : les unités de traitement et les dispositifs d'entrée-sortie (Tableau 1).

Classe des systèmes	Description
CPU	des Unités de Traitement Matérielles-Logicielles
D	des Dispositifs d'entrée-sortie

Tableau 1. *Une classification des systèmes*

On remarque que les dispositifs *D* sont les seuls moyens pour les systèmes informatiques d'interagir avec leur environnement. Nous proposons donc une classification plus spécifique de ces dispositifs dans le tableau suivant :

Catégories de dispositif	Utilisation
D_c	communication entre CPUs constituant nos réseaux de télécommunications
D_i	stockage de l'information
D_u	interaction avec l'utilisateur
D_e	entrée-sortie vers l'environnement physique

Tableau 2. *Les catégories de dispositifs*

On remarque également que les dispositifs d'entrée–sortie (D_e) vers l'environnement physique se caractérisent par l'ensemble des données qu'ils véhiculent et qui ne sont pas utilisées dans le cadre des communications entre unités de traitement, organes de stockage d'informations ou à des fins d'interactions avec l'utilisateur.

Nous retrouvons à partir de cette classification un certain nombre de thématiques de recherche qui se sont destinées à l'étude de sous-classes de systèmes mobiles. Par exemple, l'étude des agents mobiles se focalise sur la mobilité de code dans un environnement composé d'une unité de traitement (CPU), de dispositifs de communication (D_c) et de dispositifs de stockage (D_i). Par ailleurs, l'étude des applications sensibles au contexte se concentre sur un environnement essentiellement composé de dispositifs d'entrée-sortie vers l'environnement (D_e). Cette thématique de recherche se rapproche des études sur la Mobilité et les IHMs qui sont plus concernées par le triptyque comprenant l'interaction avec l'utilisateur : CPU, D_u, D_e. Étudions maintenant les liaisons qui peuvent exister entre ces entités afin de caractériser les conséquences de la mobilité de chacune d'elles sur la représentation du système complet. Une liaison, au sens informatique du terme, se traduit naturellement par l'accès (programmé dans le cas logiciel) aux fonctionnalités d'une entité par une autre. Nous pouvons alors distinguer les liaisons permanentes (stationnaires) des liaisons temporaires (in-stationnaires). L'impact de la mobilité sur les systèmes informatiques se traduit alors par l'introduction de *liaisons temporaires* qui est la conséquence directe du déplacement spatial.

2.2.*Enjeux*

Nous n'avons pas la prétention de définir, ni d'énumérer les problématiques de recherche soulevées par ces nouveaux domaines. Nous désirons simplement montrer qu'il s'agit de systèmes « multi-dispositifs », c'est-à-dire, des systèmes dont le dispositif est l'élément de construction de base et qu'à ce titre, l'Informatique Mobile devrait fournir une nouvelle génération d'environnements pour l'amélioration du développement de ces applications (Jansen *et al.,* 2005). Dans les deux cas d'Informatique Mobile ou Ambiante, la configuration dynamique du système repose sur les liaisons temporaires qui peuvent exister entre les unités de traitement de l'information (CPU) et les dispositifs (D) présents dans leur *voisinage*. Cette approche part du principe que toute application peut avoir temporairement accès à un grand nombre de dispositifs et d'unités de traitement et nécessite une gestion dirigée par l'application ou par des services dédiés visant à limiter ces accès (selon l'autorisation d'accès, l'ergonomie, la localisation, etc.). Ces concepts nous semblent plus faciles à appréhender dans la définition même des liaisons. En effets, les liaisons temporaires entre dispositifs et unités de traitement ne sont alors plus soumises aux seules capacités de communication et d'interaction mais aussi à la validation de conditions contextuelles qui autorisent ou non des unités de traitement et des dispositifs à collaborer. Nous les appelons les *liaisons contextuelles*.

Nous représentons donc un système Informatique Mobile et Ambiant comme un ensemble d'entités (unités de traitement ou dispositifs d'entrée-sortie) entre lesquelles vont se créer des liaisons contextuelles. Nous nous intéressons plus particulièrement aux systèmes d'interaction. Il reste cependant à caractériser ce qui permettra

l'établissement de ces liaisons contextuelles et donc de caractériser ce qu'est un contexte.

3.Modélisation et Représentation du contexte

Le contexte est malheureusement une notion très, voire trop, largement utilisée par de nombreuses communautés scientifiques. La conférence internationale et interdisciplinaire née en 1997 sur la Modélisation et l'Utilisation du Contexte dans des Applications (Springer-Verlag, 1997) le confirme, s'il en était besoin. Elle regroupe régulièrement des recherches sur ce thème dans des domaines aussi divers et variés que l'Intelligence Artificielle, les Sciences Cognitives, la Linguistique, la Philosophie, la Psychologie et l'Informatique Ambiante.

3.1.*État de l'art*

Parmi les systèmes Informatiques Mobiles et Ambiants tels que nous les avons décrits dans la première partie, nous nous intéressons au contexte d'interaction entre le système d'information, l'utilisateur et son environnement. Dans les premiers travaux du domaine, le contexte était souvent associé à la localisation dans l'espace dans lequel tout système mobile est naturellement appelé à évoluer (Schilit *et al.*, 1994). Néanmoins, si nous voulons dépasser la notion de localisation, la sémantique du contexte à prendre en compte se trouve alors extrêmement étendue. Ainsi plusieurs auteurs comme (Pascoe, 1998), (Dey *et al.*, 2000), (Chen *et al.*, 2000) ou (Coutaz *et al.*, 2002) ont proposé une classification des types de contexte en plusieurs familles : contexte environnemental, contexte-utilisateur, contexte-machine et contexte temporel. La notion de contexte d'interaction devient donc très large et désigne « *toute information qui peut être utilisée pour caractériser la situation des entités (c'est-à-dire une personne, un emplacement et un objet) qui sont considérées comme appropriées à l'interaction entre un utilisateur et une application, incluant l'utilisateur et l'application* », selon (Dey, 2001).

D'autres travaux (Jang *et al.*, 2003) visent à généraliser la notion de contexte en définissant la notion de contexte unifié, contrée sur une classification 5W1H (« *Who, What, Where, When, How and Why* »). Cette approche définit, notamment avant tout traitement, la notion de contexte préliminaire comme un ensemble d'éléments et d'attributs XML selon un schéma bien défini (Figure 1).

```
…
<Who>
   <Name UID="731219-xxxxx" Uncertainty="10">Seiie Jang</Name>
</Who>
<Where>
   <Location Type="Indoor">
      <Coordinates Granularity="80cm" Origin="Door"
   Uncertainty="60"><X>3</X><Y>12</Y></Coordinates>
      <Symbols Reference="ubiHome">TV</Symbols>
```

```
    </Location>
  </Where>
  <When>
    <Interval Type="Absolute" Uncertainty="20">
      <From>200504122130</From><To>200504122131</To>
    </Interval>
  </When>
  <What>
    <Destination Type="Object">
      <Identity Type="MultiMedia" Uncertainty="30">TV</Identity>
    </Destination>
  </What>
```

Figure 1. *Exemple de schéma XML.*

Ces travaux sont probablement les travaux les plus aboutis et proposent une description unifiée du contexte facilitant la manipulation des informations contextuelles au lieu d'utiliser des informations hétérogènes. En effet, avant d'être exploitées, les transformations des informations contextuelles font appel à des bases de connaissances difficilement distribuables qui s'appuient sur des ontologies, des techniques du Web sémantique (Euzenat, 2002) (Abel *et al.,* 2004) (Chen *et al.,* 2003) notamment pour la fusion et alignement d'ontologies et à des techniques d'Intelligence Artificielle (Serafini *et al.*, 2004). Dans notre approche, nous partons du principe qu'une représentation unifiée du contexte, partagée par nos différentes entités, est disponible, après traitement si nécessaire d'informations logiques ou physiques (Hofer *et al.*, 2003), externes ou internes (Prekop *et al.*, 2003), c'est-à-dire mesurées par des capteurs ou fournies par le système lui-même.

Par contre (Jang *et al.*, 2003) comme bon nombre d'autres travaux, ne font pas une distinction suffisamment claire entre ce qui nous apparaît comme définissant l'état d'une entité (*situation*) et son *contexte*. Le contexte représente le sous-ensemble des entités qui se situent au voisinage de l'entité de références. Cette nuance est clairement introduite dans les travaux de Pauty, Couderc et Banâtre (Pauty *et al,* 2004) qui ont proposé une définition plus formelle du contexte à base de fonctions de distance.

3.2. *Notre approche*

Nous souhaitons clairement distinguer les notions de *situation* et de *contexte*. En effet, la *situation* d'une entité est l'ensemble des attributs du contexte qui caractérisent son état dans un ensemble d'espaces $\{\varepsilon_i\}$. Par exemple, un espace ε_i peut être un espace classique de dimension 2 ou 3 pour la localisation GPS ou un espace plus insolite caractérisant l'état émotionnel de l'utilisateur. Une représentation vectorielle équivalente à la représentation unifiée 5W1H en XML est alors utilisée pour caractériser la situation de chaque entité. La situation d'une entité est alors décrite par un vecteur dans un espace global ε :

$$\varepsilon = \bigcup \varepsilon_i$$

Une entité e_i appartient au *contexte* d'une entité de référence e_r dans un espace donné ε_i si et seulement si la situation de e_i est dans le *voisinage* ou à *proximité* de la situation de e_r dans cet espace. Tout contexte sera alors noté $C(e_r, \varepsilon_i)$ et défini comme l'ensemble des e_i ayant vérifié la propriété précédente. Nous dirons alors qu'il existe une *liaison contextuelle* dans l'espace considéré entre les entités e_i et e_r. Nous devons donc nous doter de notions plus formelles de *proximité* et de *voisinage* maintenant inhérentes à la notion de contexte pour un espace ε_i donné. La différence entre la notion de *voisinage* et de *proximité* réside dans le type de critère retenu.

Voisinage. Si nous faisons référence à la notion de *voisinage* telle qu'elle a été mathématiquement définie dans un espace topologique (Pauty *et al,* 2004), elle est définie par la notion de distance dans l'espace ε_i (noté $d\varepsilon_i$). On peut alors écrire la définition suivante (illustrée à la Figure 2) :

$C(e_r,\varepsilon_i)=\{ e_i / d\varepsilon_i(e_i, e_r) \leq D \}$ où D est une constante.

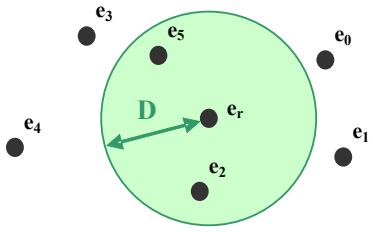

Figure 2. *Exemple dans un espace de dimension 2 muni d'une distance euclidienne.*

Proximité. Si nous faisons référence à la notion de *proximité* qui est moins contraignante et qui nous est apparu comme plus pertinente dans de nombreux cas de gestion du contexte (Lavirotte *et al.,* 2005), alors la notion de contexte $C(e_r, \varepsilon_i)$ s'appuie sur une fonction de coût dans l'espace ε_i (noté $c\varepsilon_i$) en lieu et place d'une fonction de distance. Cette fonction de coût vérifie les propriétés suivantes :

$$\begin{cases} c(x,y) \geq 0 \\ c(x,y) \neq 0 \Rightarrow x \neq y \\ c(x,y) \leq c(x,z) + c(z,y) \end{cases}$$

$C(e_r,\varepsilon_i)=\{ e_i / c\varepsilon_i(e_i, e_r) \leq C \}$ où C est une constante.

Notre modélisation du contexte nous permet donc de concevoir des filtres de conditions contextuelles dans le cadre de notre architecture. Ces filtres ont pour objectif de valider selon le contexte l'applicabilité de modifications de l'application logicielle selon des *schémas contextuels* (Figure 4). Ainsi, en fonction de la situation

des entités e_i impliquées dans une modification particulière de l'application logicielle, chaque filtre vérifie si ces entités font partie du contexte de l'entité de référence e_r au regard des conditions posées dans un espace ε_i donné. L'ensemble des modifications de l'application basée sur l'entité e_r, ensemble correspondant à un schéma contextuel centré sur e_r, est alors applicable si et seulement si l'ensemble des entités e_i impliquées passent avec succès tous les filtres. Cette sélection étant établie, nous définissons maintenant l'approche qui nous permet leur mise en œuvre.

4. Intégration et adaptation

4.1.*État de l'art*

De nombreux travaux proposent une prise en compte du contexte utilisateur dans une application logicielle, une prise en compte cependant intégrée de manière ad hoc en redéfinissant pour chaque nouvelle application ce qui constitue le contexte (Want *et al.*, 1992) (Abowd *et al.*, 1997).

Décomposition en couches. D'autres travaux essaient d'établir des principes architecturaux afin de faciliter la prise en compte du contexte dans la conception d'une application. Dans tous les cas, ces architectures font l'objet d'une décomposition en couches plus ou moins nombreuses. Au minimum, nous retrouvons un premier *niveau d'acquisition d'informations contextuelles*, un second niveau de *gestion du contexte* et un troisième niveau, *l'application* (Figure 3).

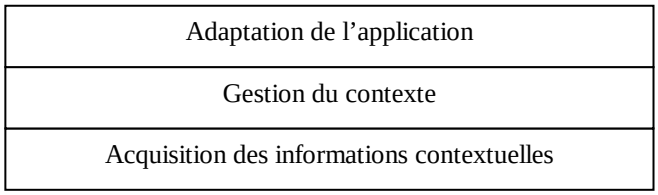

Figure 3. *Architecture en couches.*

(Chen, 2002) scinde le niveau d'acquisition des informations contextuelles requises selon les méthodes d'accès aux capteurs physiques : accès facilité par une infrastructure de type intergiciel ou accès obtenu depuis un serveur de contexte centralisé. (Winograd, 2001) propose un classement selon trois familles de gestion du contexte : les plates-formes de Widgets, les réseaux de services et les blackboards.

Modèles architecturaux. Au regard de la littérature du domaine, nombreux sont les modèles d'architectures logicielles mis en œuvre dans le cadre de la conception et du développement d'applications sensibles au contexte. Nous pouvons citer la Context Toolkit (Dey *et al.*, 2001) qui s'apparente à un intergiciel pour le contexte,

des approches basées sur la notion de Contexteur (Coutaz *et al.*, 2002) ou d'objets sensibles (sous-entendu au contexte) de CORTEX (Biegel *et al.*, 2004) qui s'apparentent à des systèmes multi-agents, Ubi-UCAM (Jang *et al.*, 2003), une architecture orientée service pour la gestion du contexte et les « Interactives Workspaces » de (Fox *et al.*, 2000) qui s'appuient sur une approche centrée sur des blackboards. Nous retiendrons que ces approches sont structurées sous forme de couches (Baldauf *et al.*, 2006).

Contexte d'exécution. Dans le domaine de la prise en compte du contexte d'exécution, l'objectif est de permettre à l'application de gérer les situations externes qui influencent sa qualité de service perçue par les utilisateurs. On peut alors parler d'Informatique Autonome (*Autonomic Computing*) présentée par IBM (Horn, 2001) où, à l'image des capacités d'auto-adaptation des systèmes biologiques, une application doit pouvoir s'adapter à toutes sortes de défaillances techniques, de malveillances et de charge inhabituelle. Il s'agit alors très souvent d'adapter l'application à l'évolution de l'état des ressources du système sur lequel elle s'exécute.

Extraction d'informations. Certains travaux fournissent alors des techniques d'extraction et de représentation des informations sur le contexte d'exécution de l'application pour qu'elles puissent être utilisées dans l'adaptation des applications. Par exemple RAJE (*Ressource-Aware Java Environment)* et SAJE (*System-Aware Java Environment)* (Mahéo *et al.*, 2004) constituent un cadre de conception définissant les fonctionnalités nécessaires à l'observation des ressources système dans le monde Java. Il offre le moyen de réifier ces ressources sous forme d'objets afin d'en observer l'état. SAJE fournit des fonctionnalités pour adapter ou superviser des ressources. (David *et al.*, 2006) modélise dans WildCAT le contexte d'exécution par un ensemble de « domaines contextuels » qui représentent chacun un aspect spécifique du contexte comme les ressources matérielles, le réseau (topologie, performances), le contexte géophysique (position géographique, température ambiante) au travers une représentation arborescente du contexte sous forme [CONTEXTE, DOMAINE, RESSOURCE, ATTRIBUT]. LeWYS (Cecchet *et al.*, 2005) est un canevas à composants dédié à la construction de systèmes d'observations. Il fournit différentes sondes permettant de réifier un vaste spectre de données provenant du système d'exploitation et des applications en cours d'exécution. Quant à COSMOS (Bellavista *et al.*, 2003), il s'agit d'un intergiciel pour la gestion de la sécurité pour des Agents Mobiles sur un réseau Internet sans fil. Le contexte et les autorisations de l'agent sont alors décrits par des méta-données à travers des profils utilisateur-équipement, nomade-ressources et des règles d'autorisation d'accès au système du type système-{niveau d'autorisation pour l'utilisateur}. Ces approches proposent souvent un fort découplage entre acquisition et représentation du contexte et la mise en œuvre des adaptations résultantes de l'application en utilisant une ou plusieurs représentations intermédiaires du contexte.

Dans le domaine de l'Informatique Mobile et Ambiante, après une période d'expérimentations entremêlant études de cas et preuves de concept, certains travaux ont cherché à synthétiser et caractériser les enjeux de la thématique. Par exemple, (Satyanarayanan, 1996) pose explicitement la question de ce qui est unique et différent conceptuellement dans l'Informatique Mobile. Il en déduit la nécessité pour ces applications d'être avant tout adaptatives et met en évidence le continuum des solutions qui peuvent exister entre les approches alors les plus usitées :

-les approches « Laissez-Faire » (adaptation gérée par l'application)

-les approches « Transparentes » (le système gère l'adaptation)

Nous pouvons ainsi introduire la notion d'intergiciel d'adaptation et distinguer différents paradigmes logiciels utilisés pour l'adaptation : la réflexivité logicielle (Maes, 1987), la conception orientée composant (Szyperski, 1998), la programmation orientée aspect (Kiczales *et al.*, 1997) et schémas de conception (Schmidt, 2000). Dans le cadre de l'étude d'une taxonomie de l'adaptation dans (McKinley *et al.*, 2004), les auteurs distinguent les différentes approches par le « comment, quand et où » les compositions logicielles à des fins d'adaptation prennent donc effet ? Cette analyse permet aux auteurs de produire un état de l'art remarquable mais qui fait néanmoins peu de cas de la complexité des interventions nécessaires pour introduire des stratégies d'adaptation. En d'autres termes, quelles sont les entités qui doivent intervenir dans l'adaptation d'une application ? Est-ce un développeur avec des outils basés sur des langages, un architecte logiciel dans l'agencement de composants, un utilisateur au travers le téléchargement d'application complète sur son terminal mobile ? La réponse est très liée aux techniques et paradigmes logiciels précités. Par exemple, Rashid et Kortuem s'appuient sur la programmation orientée aspect pour adapter des applications aux environnements pervasifs (Rashid *et al.*, 2004). Cette approche est particulièrement adaptée à la modification transversale des propriétés et du comportement d'une application. En ce sens, il peut s'agir d'une approche efficace pour l'adaptation d'une application, même si celle-ci demeure centrée sur un langage de programmation et reste destinée au développeur.

4.2.*Notre approche*

Pour la prise en compte du contexte et l'adaptation de l'application, notre approche repose sur la modification dynamique d'une application résultante d'un assemblage de composants. Même si un tel modèle de conception reste compatible à une description en couches de l'application nous préférons mettre l'accent sur des mécanismes de modifications transverses (*cross-layered*) d'assemblage de composants en exploitant le principe de séparation des domaines de préoccupation (Figure 4) dans la notion de schéma contextuel.

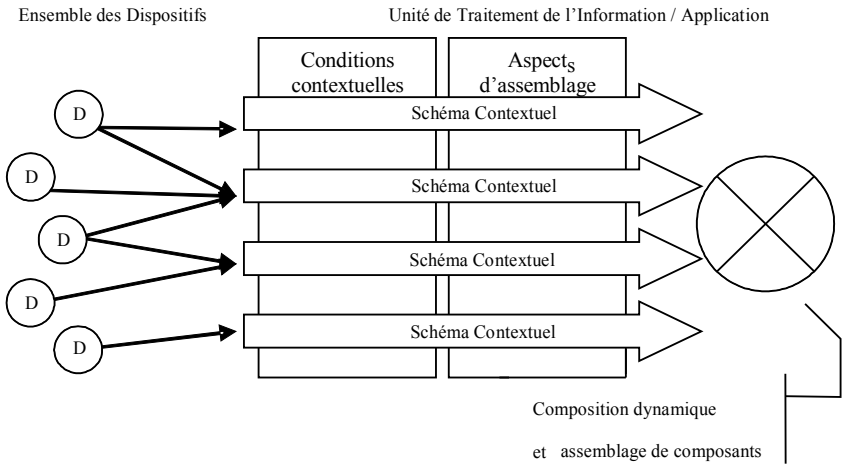

Figure 4. *Approche « cross-layered »*

Nous travaillons sur l'adaptation de l'application à des contextes selon un domaine particulier tel que la sécurité, la localisation, l'état émotionnel de l'utilisateur, etc. Les *schémas contextuels* sont composés de deux parties : la première vérifiant les conditions contextuelles d'application des adaptations et la deuxième spécifiant ces adaptations sous forme d'un aspect d'assemblage.

```
schéma contextuel = <conditions contextuelles, aspect d'assemblage>
```

Nous nous appuyons sur une technique d'application simultanée de plusieurs aspects d'assemblage pour une application écrite sous forme d'un assemblage de composants. Nous pouvons trouver ici une certaine similitude avec la démarche des règles actives ECA (Bounaas, 1995). Une application peut être décrite comme un ensemble de règles actives où chaque règle est définie sous la forme d'un *t*-uplet Évènement-Condition-Action (ECA). L'exécution de l'application consiste alors à détecter les évènements, évaluer les conditions et lancer les actions correspondantes. Les nuances de notre approche reposent sur la distribution des informations contextuelles qui ne sont pas nécessairement centralisées dans une base de données et la combinaison logique (ou fusion) des l'assemblage de composants sous forme d'aspects d'assemblage (cf. section 5.2).

5.Adaptation en utilisant des composants et des aspects d'assemblage

Notre approche pour la prise en compte du contexte consiste donc en l'application de schémas contextuels. Chaque schéma contextuel définit un aspect d'assemblage sélectionné en fonction de conditions contextuelles.

Nous présenterons dans une première section la plate-forme Wcomp et la modélisation des composants logiciels associée, puis le modèle ISL4Wcomp et la notion d'aspects d'assemblage. Nous voyons ensuite comment utiliser les aspects d'assemblage de Wcomp décrit dans des schémas contextuels. Finalement, nous présentons quelques exemples de schémas, de conditions contextuelles et d'aspects d'assemblage associés.

5.1.*La plate-forme Wcomp*

La plate-forme Wcomp (Cheung-Foo-Wo *et al.*, 2006c) est un projet qui a pour but de mettre en place une plate-forme de développement rapide pour les systèmes dynamiques multi-dispositifs. Elle offre un environnement de développement à base de composants logiciels. La vocation de Wcomp est de distinguer deux catégories de composants : les *composants logiciels* (**component**) « libres » et les *composants mixtes* (**mixed component**) c'est-à-dire des composants logiciels dépendants d'un ou plusieurs dispositifs d'entrée/sortie. Un composant est doté d'entrées et de sorties. Il est possible de modifier les connexions de ces entrées-sorties. La reconfiguration dynamique sur Wcomp consiste à ajouter, retirer, connecter ou déconnecter des composants logiciels pendant l'exécution d'une application.

5.1.1Modèle de composant

Un composant dans Wcomp est l'instance d'une classe qu'il soit implémenté en Java, C#, Objective-C ou C++. Un composant est muni de *ports d'entrée* (input) qui sont des méthodes et de *ports de sortie* (output) qui sont des évènements qui peuvent être émis.

Un évènement est donc une diffusion de messages dynamiques et contrôlables par l'application elle-même. Ces messages déclenchent autant d'appels de méthodes abonnées à l'événement correspondant. Dans le cas où leurs signatures ne sont pas compatibles, il s'agit de pouvoir combler les paramètres manquants de la méthode cible. La signature de la méthode devant contenir la signature du message, la valeur d'un paramètre manquant peut-être obtenue par l'appel d'une méthode complémentaire sur un composant défini. Bien entendu, le type de la valeur de retour de la méthode complémentaire doit être le même que celui du paramètre manquant

Le modèle de composants Wcomp en s'inspirant très fortement du modèle JavaBeans (Hamilton, 1997) n'a donc pas pour ambition de se distinguer des autres approches à composants (Pessemier *et al.*, 2004) (Gay *et al.*, 2003). Son principal objectif est de fournir un modèle particulièrement adapté à l'utilisation de dispositifs d'entrée-sortie.

Il privilégie pour cela des communications par diffusion d'événements et se distingue ainsi de la notion d'interface définie en Fractal. Une *interface requise* en Fractal est un ensemble de fonctions et d'appels de fonction dont un composant doit disposer afin de réaliser sa fonctionnalité correctement. Une *interface fournie* est un ensemble de fonctions et d'appels de fonction qu'un composant offre afin de réaliser un certain service pour un composant externe. En revanche, en Wcomp, une fonction représente *l'entrée* d'un composant et une diffusion d'évènement représente une *sortie*.

5.1.2Architecture

Une *application* dans Wcomp consiste ainsi en un ou plusieurs composants interconnectés afin de former un programme exécutable, appelé *assemblage de composants.* L'outil gérant l'assemblage est appelé *Container.*

En complément nous appelons les outils pour manipuler ces composants, les instancier et les interconnecter : les *Designers* (Cheung-Foo-Wo *et al.*, 2006a et 2006c). Nous avons un *designer* pour chaque représentation de l'application. Dans cet article, nous allons détailler le *designer* intitulé « *designer ISL4Wcomp* ». Ce *designer* permet de manipuler l'application sous la forme de *composition d'aspects d'assemblage* (Cheung-Foo-Wo *et al.*, 2006b).

La communication entre un *designer* et un *container* se fait à partir d'une interface de programmation composée de cinq fonctions principales : ajout et retrait de composants, liaison et rupture de liaison et obtention de l'assemblage. Cette structure permet de travailler sur une même application à partir de plusieurs *designers*.

La modification d'une application se conçoit alors comme un *réassemblage* de ses composants au travers un designer. Afin d'effectuer le réassemblage dynamique d'applications à base de composants, nous nous appuyons sur les commandes de modifications élémentaires de l'assemblage définies dans (Cheung-Foo-Wo *et al.*, 2006a) qui sont :

-*add_component Type (Id)* [ou simplement *add*]

-*remove_component Id* [ou simplement *remove*]

-*link Id Source Event Target Method (Params)*

-*unlink Id*

Ces fonctions sont appelées *modifications élémentaires* d'une application.

Prenons un exemple pour illustrer son utilisation. On souhaite abonner une méthode m_2 de signature *void* (t_1, t_2, t_3) à l'émission de l'événement $\hat{}m_1$ de signature *void* (t_1, t_2). Les t_1, t_2 et t_3 sont des types. On constate que la méthode m_2 n'est pas complètement compatible avec $\hat{}m_1$. Supposons qu'il existe une troisième méthode m_3 de signature t_3 (). La valeur de retour de m_3 correspond au paramètre manquant. On peut donc abonner la méthode m_2 à l'événement $\hat{}m_1$ en effectuant :

link (^m_1, c_1, m_2, c_2, m_3, c_3)

L'usage des langages de description d'architectures logicielles ADL (Medvidovic *et al.*, 2000) pour décrire les assemblages est aujourd'hui largement répandu. Une autre approche telle que celle utilisée dans nesC (Gay *et al.*, 2003) permet au développeur une description plus libre de l'assemblage au travers la programmation de connecteurs complexes appelés « configurations » en complément des composants appelés « modules ». Dans tous les cas, ces approches ne prennent pas en charge la superposition d'assemblages selon une sémantique explicite basée sur des règles logiques telle que nous la gérons par l'intermédiaire de tissage d'aspects.

Après la plate-forme Wcomp, nous présentons donc plus particulièrement le *designer ISL4Wcomp* permettant de gérer les modifications apportées à l'application par tissage d'aspects d'assemblage de composants.

5.2.*Aspects d'assemblage : ISL4Wcomp*

Notre technique d'application des aspects d'assemblage est constituée d'un mécanisme de *tissage d'aspects* dans le domaine de la programmation logicielle orientée aspect AOP appliquée aux composants logiciels (Seinturier, 2006) pour modifier l'assemblage qui implémente l'application. Les aspects d'assemblage sont décrits en respectant les interfaces des composants Wcomp. La composition des aspects est réifiée au niveau d'un autre modèle de l'application : le modèle ISL. Le calcul de leur tissage et l'évolution dynamique de l'assemblage de composants sont pris en charge au niveau de ce modèle. Le résultat des calculs est ensuite projeté en termes de réassemblages des composants Wcomp. Il s'agit d'une des particularité de notre approche sur la fusion des aspects d'assemblage (Cheung-Foo-Wo *et al.*, 2006b).

Applicabilité. On modélise un aspect par une fonction qui retourne une liste de modifications élémentaires et qui prend comme paramètres une liste (d_1, …, d_n) de descriptions de composants représentant les composants. Soit O l'ensemble des listes de modifications élémentaires, on a :

$$s_{e_r} \begin{cases} D^n \rightarrow O \cup \{nil\} \\ (d_1, \ldots, d_n) \mapsto \text{commandes} \end{cases}$$

où $o \in O$ si le schéma est applicable

Un aspect s_{er}=(d_1, …, d_n) est applicable par rapport à une entité de référence e_r, si pour chaque entité e_i correspondant aux composants d_i qui sont mixtes, tous les e_i sont dans la zone contextuelle de e_r.

Transformation aspect-réassemblage. A chaque notification de modification du contexte d'une entité e_r, pour chaque aspect d'assemblage s_{er} on considère les cas suivants :

-s_{er} est déjà posé et n'est plus applicable,

-s_{er} n'est pas posé et est applicable.

Dans le premier cas, on « défait » les modifications opérées par s_{eri} (si celui-ci existe) lors de la précédente transformation en fournissant les modifications élémentaires afin de défaire les modifications. Dans le second cas, on exécute les modifications élémentaires retournées par s_{er}. Pour les autres cas, il n'y a aucune action à entreprendre. Nous identifions toutefois un problème lié à cette modélisation. Défaire des modifications opérées par la transformation d'un aspect d'assemblage en une liste de modifications élémentaires est complexe. Il peut être dépendant de l'historique de tous les aspects qui ont été posés auparavant.

Comment écrit-on un aspect d'assemblage ? Nous nous appuyons sur le langage de spécification d'interactions ISL (Berger, 2001). ISL (*Interaction Specification Language*) est un langage de description de schémas d'interactions entre des composants. Ce langage a la propriété d'être *composable* c'est-à-dire que plusieurs schémas écrits en ISL peuvent se composer et fusionner en un seul schéma combinant leur comportement. Nous retrouvons les propriétés d'ISL dans les aspects d'assemblage ISL4Wcomp, notamment la symétrie – c'est-à-dire que l'ordre d'application des aspects n'influence pas le résultat de leur composition – démontrée par Berger. Au sein d'ISL4Wcomp, nous avons défini deux façons de modifier le comportement d'une application : d'une part, sur un appel de méthode, d'autre part, sur la diffusion d'un évènement, dans tous les cas, en y associant un nouveau comportement programmé avec les opérateurs du langage (cf. Tableau 3).

```
<point de jonction> : <composant>.<évènement> | <composant>.<méthode>

aspect <nom> (<composant>…) {
     <point de jonction> { <comportement programmé ISL4Wcomp> } }
```

Un aspect d'assemblage repose donc sur des ensembles de règles. Une règle porte sur des points de jonctions : émission d'un évènement ou appel de méthode.

Opérateurs	Description
;	Exprime une séquence
||	Exprime un ordre indifférent
if … else …	Exprime une condition
call	Désigne l'événement ou l'appel de méthode du point de jonction
delegate	Désigne l'émission d'un autre événement ou l'appel d'une autre méthode.
^	Exprime l'émission d'un évènement

Tableau 3. *Principaux opérateurs du langage ISL4Wcomp*

Certains mots-clés (cf. Tableau 3) du langage comme *call* et *delegate* permettent de contrôler la manière dont les aspects d'assemblage vont se composer. Nous pouvons les rapprocher des mots-clés comme *before*, *around* et *after* d'AspectJ.

La fusion des aspects d'assemblage correspond à la mise en œuvre des travaux formels sur la composition des règles ISL, étendus à ISL4Wcomp. La définition d'aspects d'assemblage est un moyen de représenter les assemblages de composants en y apportant la possibilité de les fusionner automatiquement. Des exemples de composition d'aspects d'assemblage peuvent être trouvés dans (Cheung-Foo-Wo *et al.,* 2006b). Nous décrivons dans le paragraphe ci-dessous l'écriture d'un aspect.

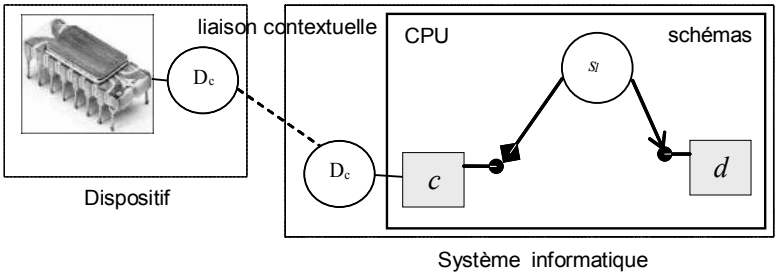

Figure 5. *Schéma expérimental : système informatique ambiant*

Exemple d'aspect. Nous définissons un aspect d'assemblage qui permet de mettre en relation plusieurs composants (Figure 5) :

```
aspect s1 (d, c) { d.^Temperature() { call || c.Temperature() } }
```

Comparaison. Parmi les travaux du domaine, les composants d'aspect Fractal FAC (Pessemier *et al.*, 2004 et 2005) interprètent d'une première manière la notion de *préoccupations transverses* c'est-à-dire les aspects logiciels. Pour Pessemier, les préoccupations doivent être introduites sous la forme d'une entité logicielle composite et de plusieurs liaisons entre ce composite et les composants de l'application. Dans ISL4Wcomp, nous introduisons le contrôle au niveau des aspects d'assemblage et non pas au niveau d'un composite. FuseJ (Suvée *et al.*, 2005) est proche de FAC en ne faisant pas de différence entre aspects et composants. FuseJ relègue cependant la notion de coupe au sein du connecteur. En ISL4Wcomp, les aspects d'assemblage sont traduits en un ensemble de composants et de liaisons qui s'intègrent à l'assemblage de l'application. Cet ensemble est considéré comme étant un aspect en FuseJ si l'aspect spécifie l'ordre dans lequel les fonctionnalités doivent être appelées par les mots-clés *before*, *after* et *replace*. En ISL4Wcomp, cet ordre dépend des opérateurs (cf. Tableau 3) utilisés dans la description des schémas et du mécanisme de fusion des opérateurs défini par les *règles de fusion ISL* (Berger, 2001). SAFRAN (David *et al.*, 2006) acronyme de « *Self-Adaptative FRActal compoNents* », s'adresse au problème de la reconfiguration dynamique d'assemblages

de composants en fonction du contexte d'exécution. On y décrit *des schémas d'adaptation* qui s'appuient sur des règles ECA. Mais SAFRAN ne met pas l'accent sur la manière dont sont pris en compte les conflits potentiels entre les règles sélectionnées. Avec ISL4Wcomp, nous proposons une combinaison logique des opérateurs du langage. Le résultat de l'application de règles va être une combinaison logique et non pas une juxtaposition des corps des règles. On se positionne non plus à un niveau hiérarchique face à l'assemblage de composants mais à un niveau fonctionnel en décrivant le comportement attendu en entrée ou en sortie de chaque composant. Ainsi, le résultat de l'application de règles en conflit est résolu de manière logique par une fusion comportementale ISL.

5.3.*Application aux schémas contextuels*

Nous employons ISL4WComp pour intégrer les composants mixtes. Cette intégration se fait en appliquant certains aspects d'assemblages au nouvel ensemble de composants. Nous décrivons dans cette partie la liaison entre les conditions contextuelles et l'aspect d'assemblages qui constitue un schéma contextuel. La sélection de l'aspect se fait à partir des informations contextuelles. Un aspect est sélectionné lorsqu'un ensemble d'entités, ici des dispositifs, se situe dans le contexte $C(e_r, \varepsilon_i)$ vérifiant les condtions contextuelles du schéma contextuel correspondant.

Nous proposons à présent de décrire un exemple simple d'application que nous avons mis en œuvre dans un laboratoire d'expérimentation pour l'Informatique Ambiante appelé « Ubiquarium Informatique » (Hourdin *et al.*, 2006).

Nous considérons six dispositifs dont un mobile avec l'utilisateur – le Joystick et un autre qui n'est pas mixte – l'écran portable ou HeadMountedDisplay (HMD):

```
mixed component Joystick;
implementation {
   output Clicked (), Down (), Up (), Left (), Right ();
}
```

```
mixed component EcranTv;
implementation { input JouerFilm() { … }, Eteindre() { … } }
```

```
mixed component ChaineHifi;
implementation {
   input JouerChanson() { … }, Eteindre() { … }
}
```

```
mixed component Magneto;
implementation { input Marche() { … }, Arrêt() { … } }
```

```
mixed component PlaquesCuisson;
```

```
implementation {
   input Allumer1() { … }, Eteindre1() { … }
   input Allumer2() { … }, Eteindre2() { … }
}
```

```
component HeadMountedDisplay;
implementation {
   output AfficherInformations(String message) ;
   input Print(msg) { … }
}
```

Dans cette application, l'entité de référence est le Joystick. Les différentes informations contextuelles et les espaces associés communs aux entités de l'application sont : l'heure du jour, l'orientation de l'entité, la position de l'entité dans la pièce. L'utilisateur se déplace avec le joystick et selon son orientation, les informations qui apparaissent sur son HMD changent d'une part, et d'autre part, les liaisons entre le Joystick et les entités environnantes se modifient.

Voici les schémas contextuels associés à chaque entité :

Schéma contextuel AEcranTV.

`Condition contextuelle` **:** L'aspect ***AEcranTV*** est applicable si et seulement si les composants *joystick* et *ecrantv* sont à proximité contextuelle c'est-à-dire que l'entité EcranTV doit être dans le champ de vision du Joystick, à une distance inférieure à un seuil donné et que la tranche horaire soit incluse dans l'ensemble {12h-14h, 19h-21h} par exemple.

```
aspect AEcranTV (joystick, hmd, ecrantv) {
   hmd.^AfficherInformations(msg) {
      call ;
      hmd.Print("Cliquer pour allumer la télé.") ;
      hmd.Print("Bas pour étendre la télé.")
   }
   joystick.^Clicked() { call || ecrantv.JouerFilm() }
   joystick.^Bas() { call || ecrantv.Eteindre() }
}
```

Description de l'aspect. La première règle de l'aspect spécifie que lorsque l'évènement AfficherInformations est émis par le composant *ecrantv*, alors on effectue le comportement prévu (comportement résultat de la composition des autres aspects) et puis on affiche successivement deux messages sur le HMD. La seconde règle signifie que lorsque l'évènement Clicked est émis par le composant joystick, alors on effectue le comportement prévu et sans définir d'ordre précis, on joue un film sur l'écran-tv. La troisième règle est similaire à la deuxième mais défini « Bas » comme évènement et « éteindre la télévision » comme action.

De la même manière, nous spécifions les aspects suivants mais sans les détailler comme précédemment.

Schéma contextuel AChaineHifi.

`Condition contextuelle :` L'aspect ***AChaineHifi*** est applicable si et seulement si les composants *joystick* et *chainehifi* sont à proximité contextuelle c'est-à-dire que l'entité *chainehifi* doit être dans le champ de vision du *joystick*, à une distance inférieure à un seuil donné et que la tranche horaire soit incluse dans l'ensemble {12h-14h, 19h-21h}.

```
aspect AChaineHifi (joystick, hmd, chainehifi) {
   hmd.^AfficherInformations(msg) {
      call ;
      hmd.Print("Cliquer pour jouer une chanson.") ;
      hmd.Print("Bas pour arrêter la chanson.")
   }
   joystick.^Clicked() { call || chainehifi.JouerChanson() }
   joystick.^Bas() { call || chainehifi.Eteindre() }
}
```

Résultat de la fusion AChaineHif + AEcranTV.

`Condition contextuelle :` Le joystick se trouve à proximité contextuelle à la fois de la chaîne Hifi et de l'écran de télévision. La tranche horaire est la même pour les deux schémas contextuels et est satisfaite.

```
aspect résultat (joystick, hmd, chainehifi, ecrantv) {
   hmd.^AfficherInformations(msg) {
      call ;
         ( hmd.Print("Cliquer pour allumer la télé.") ;
           hmd.Print("Bas pour étendre la télé.") )
      || ( hmd.Print("Cliquer pour jouer une chanson.") ;
           hmd.Print("Bas pour arrêter la chanson.") )
   }
   joystick.^Clicked() { call || chainehifi.JouerChanson()
                              || ecrantv.JouerFilm() }
   joystick.^Bas() { call || chainehifi.Eteindre()
                          || ecrantv.Eteindre() }
}
```

Le résultat de la fusion consiste à affecter la fonctionnalité Clicked du joystick à la fois à la fonction « Jouer une chanson » sur la chaîne et « Jouer un film » à l'écran. Ces deux fonctionnalités sont effectuées en parallèle selon la fusion ISL. Ceci dit, la sémantique du parallélisme est déléguée à la plate-forme d'exécution. Si le comportement attendu n'est pas celle du parallélisme telle qu'elle est décrite en ISL, nous avons besoin d'un modèle de la gestion des ressources pour pouvoir décider de sa sémantique finale.

Schéma contextuel AMagneto.

`Condition contextuelle :` L'aspect ***AMagneto*** est applicable si et seulement si les composants *joystick* et *ecrantv* sont à proximité contextuelle c'est-à-dire que l'entité *ecrantv* doit être dans le champ de vision du Joystick et à une

distance inférieure à un seuil donné. Aucune condition n'est fixée sur la tranche horaire.

```
aspect AEcranTV (joystick, hmd, magneto) {
   hmd.^AfficherInformations(msg) {
      delegate {
         hmd.Print("Cliquer pour lancer le magneto.") ;
         hmd.Print("Bas pour éteindre le magneto.") }
   }
   joystick.^Clicked() { delegate magneto.Marche() }
   joystick.^Bas() { delegate magneto.Arret() }
}
```

Schéma contextuel APlaquesCuisson.

Condition contextuelle : L'aspect ***APlaquesCuisson*** est applicable si et seulement si les composants *joystick* et *plaques* sont à proximité contextuelle c'est-à-dire que l'entité *plaques* doit être dans le champ de vision du Joystick, à une distance inférieure à un seuil donné et que la tranche horaire soit incluse dans l'ensemble {12h-14h, 19h-21h}.

```
aspect APlaquesCuisson (joystick, hmd, plaques) {
   hmd.^AfficherInformations(msg) {
      delegate {
         hmd.Print("Bas pour allumer la plaque 1.") ;
         hmd.Print("Haut pour éteindre la plaque 1.") ;
         hmd.Print("Gauche pour allumer la plaque 2.") ;
         hmd.Print("Droit pour éteindre la plaque 3.") }
   }
   joystick.^Bas() { delegate plaques.Eteindre() }
   joystick.^Haut() { delegate plaques.Eteindre() }
   joystick.^Gauche() { delegate plaques.Eteindre() }
   joystick.^Droit() { delegate plaques.Eteindre() }
}
```

Discussion. Par définition d'aspect d'assemblage, nous avons la possibilité de composer plusieurs aspects tels que décrits ci-dessus afin d'obtenir une application globalement cohérente. Nous avons mis en évidence un point de cohérence entre aspects : leur priorité. L'utilisation du mot-clé *delegate* permet d'indiquer que le comportement spécifié est prioritaire par rapport à un comportement utilisant juste un *call*. Ainsi par exemple, si le joystick se situe à la fois dans le contexte des plaques de cuisson et celui de la chaîne hifi, alors ce sont les plaques de cuisson qui seront contrôlées et non pas la chaîne hifi. Cependant, nous avons uniquement la possibilité de définir deux niveaux de priorité qui proviennent de la définition même des spécifications du langage ISL. En effet, ISL permet d'agencer de façon cohérente la composition de séquences et de concurrences afin de n'inculquer aucun ordre entre les opérations lorsque cela n'est pas spécifié explicitement par un des aspects. Cependant, ISL définissait la composition de deux délégations comme une erreur. Nous travaillons sur un ordonnancement correct de la composition à plusieurs niveaux de priorité.

6.Conclusion

L'adaptation dynamiquement d'une application logicielle à un environnement d'exécution découvert dynamiquement, évoluant tout aussi dynamiquement, et partiellement connue à priori n'est pas chose aisée. Toutefois, le paradigme qui permet de gérer une telle application par assemblage de composants se révèle alors particulièrement pertinent notamment associé à un langage d'aspect pour la composition dynamique d'une part et un ensemble de composants orientés services pour la découverte dynamique de dispositifs d'autre part. En fonction des contextes, et notamment des dispositifs disponibles, différents aspects peuvent alors être sélectionnés, appliqués et tissés pour adapter l'application. WComp est un environnement de prototypage d'applications sensibles au contexte basé sur ces concepts.

En conclusion, l'Informatique Mobile présente deux caractéristiques nouvelles. A l'image des ordinateurs portés, les systèmes informatiques mobiles doivent gérer *l'hétérogénéité* (Oprescu, 2004) de leurs composants informatiques, hétérogénéité induite par la très grande diversité des dispositifs utilisables et utilisés. Ils doivent aussi aborder l'utilisation alternative de dispositifs non portés par l'utilisateur. A l'image de l'Informatique Ambiante les systèmes informatiques mobiles sont aussi affectés par la disponibilité variable des dispositifs utilisables, soit la *dynamicité* de l'ensemble de leurs composants informatiques. Nous avons proposé une modélisation des interactions entre le système informatique, l'utilisateur et son environnement, et nous avons proposé une approche pour la prise en compte du contexte pour le déclenchement de l'adaptation logicielle par tissage d'aspects d'assemblage de composants déclenchés par des conditions contextuelles. L'originalité de notre approche repose alors sur deux apports :

- la définition et l'utilisation d'un contexte unifié pour la mise en œuvre de filtres de conditions contextuelles validant le déclenchement d'adaptations
- une description « cross-layered » ou transversale de ces adaptations sous forme d'aspects d'assemblage composables.

Les mécanismes décrits sont implémentés à l'aide de Wcomp et ISL4WComp et expérimenté dans l'« Ubiquarium Informatique » que nous mis en place.

7.Bibliographie

Abel F., Brase J., Using Semantic Web Technologies for context-aware Information Providing to Mobile Devices, Technical report, Institute Knowledge Based Systems, Hanover, Germany, 2004.

Abowd G. D., Atkeson C. G., Hong J., Long S., Kooper R., Pinkerton M., « Cyberguide : A Mobile Context-Aware Tour Guide », *Baltzer/ACM Wireless Networks*, vol. 3, p. 421-433, 1997.

Baldauf M., Dustdar S., Rosenberg F., « A Survey on Context-Aware systems », *International Journal of Ad Hoc and Ubiquitous Computing, forthcoming*, 2006.

Bellavista P., Montanari R., Tibaldi D., « COSMOS : A Context-Centric Access Control Middleware for Mobile Environments », *MATA : mobile agents for telecommunication applications*, vol. 2881, p. 77-88, October, 2003.

Berger L., Mise en Oeuvre des Interactions en Environnements Distribués, Compilés et Fortement Typés : le Modèle MICADO, Thèse de doctorat, Université de Nice - Sophia Antipolis, Octobre 2001.

Biegel G., Cahill V., « A Framework for Developing Mobile, Context-aware Applications », *Proceedings of the Second IEEE conference on Pervasive Computing and Communications*, IEEE Computer Society, Washington, DC, USA, p. 361, 2004.

Bounaas F., « Using ECA Rules for Object and Schema Evolution in an Object-Oriented System », *Technology of Object-Oriented Languages and Systems*, USA, 1995.

Cecchet E., Layaïda O., Quéma V., « LeWYS : un Canevas Logiciel à Composants pour Construire des Applications de Supervision », *Journées sur les Systèmes à Composants Adaptables et Extensibles*, Le Croisic, France, Avril, 2005.

Cépaduès (ed.), *Actes des Premières Journées Ubiquité Mobilité*. ACM, Nice Sophia-Antipolis, France, 2004.

Chen G., Kotz D., A Survey of Context-Aware Mobile Computing Research, Technical report, Dept. of Computer Science, Dartmouth College, 2000.

Chen H., An Intelligent Broker Architecture for Context-Aware Systems, Technical report, University of Maryland Baltimore County, December, 2002.

Chen H., Finin T., Joshi A., « An Ontology for Context-Aware Pervasive Computing Environments », *Knowledge Engineering Review*, vol. 18, n°3, p. 197-207, 2003.

Cheung-Foo-Wo D., Blay-Fornarino M., Tigli J.-Y., Lavirotte S., Riveill M., « Adaptation dynamique d'assemblages de dispositifs par des modèles », *2ème Journées sur l'Ingénierie Dirigée par les Modèles (IDM)*, 2006a.

Cheung-Foo-Wo D., Blay-Fornarino M., Tigli J.-Y., Dery A.-M., Emsellem D., Riveill M., « Langage d'aspect pour la composition dynamique de composants embarqués », *RTSI -L'Objet*, vol. 12, n° 2-3, p. 89-111, 2006b.

Cheung-Foo-Wo D., Tigli J.-Y., Lavirotte S., Riveill M., « Wcomp: a Multi-Design Approach for Prototyping Applications using Heterogeneous Resources », *17th IEEE International Workshop on Rapid System Prototyping*, Chania, Crete, June, 2006c.

Coutaz J., Rey G., « Recovering foundations for a theory of contextors », *4th International Conference on Computer-Aided Design of User Interfaces*, Valenciennes, France, p. 283–302, Mai, 2002.

David P.-C., Ledoux T., « Une approche par aspects pour le développement de composants Fractal adaptatifs », *RSTI - L'Objet*, vol. 12, n° 2-3, p. 113-132, 2006.

Dey A. K., « Understanding and using context », *Personal and Ubiquitous Computing*, vol. 5, n° 1, p. 4-7, 2001.

Dey A. K., Abowd G., « Towards a better understanding of context and context-awareness », *Workshop on the What, Who, Where, When, and How of Context-Awareness*, The Hague, The Netherlands, Juin, 2000.

Dey A. K., Salber D., Abowd G., « A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications », *Human Computer Interaction J.*, vol. 16, p. 97–166, 2001.

Euzenat J., « Research challenges and perspectives of the Semantic web », *IEEE Intelligent Systems*, vol. 17, n° 5, p. 86-88, October, 2002.

Fox A., Johanson B., Hanrahan P., Winograd T., « Integrating information appliances into an interactive workspace », *IEEE Computer Graphics and Applications*, vol. 20, n° 3, p. 54-65, June, 2000.

Gay D., Levis P., von Behren R., Welsh M., Brewer E., Culler D., « The nesC Language : A Holistic Approach to Networked Embedded Systems », *Proceedings of Programming Language Design and Implementation (PLDI'03)*, 2003.

Hamilton G., JavaBeans, Sun Microsystems, API Specification, Version 1.01, July, 1997.

Hofer T., Schwinger W., Pichler M., Leonhartsberger G., Altmann J., Retschitzegger W., « Context-Awareness on Mobile Devices - the Hydrogen Approach », *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, 2003.

Horn P., Autonomic Computing: IBM's Perspective on the State of Information Technology, Technical report, IBM Research Division, October, 2001.

Hourdin V., Cheung-Foo-Wo D., Lavirotte S., Tigli J.-Y., « Ubiquarium Informatique : Une plate-forme pour l'étude des usages des équipements informatiques mobiles en environnement simulé », *Poster pour les 3èmes Journées Francophones Mobilité et Ubiquité (UbiMob'06)*, Septembre, 2006.

Jang S., Woo W., « Ubi-UCAM : A Unified Context-Aware Application Model », *Context'03*, vol. 2680, p. 178-189, 2003.

Jansen E., Abdulrazak B., Yang H., King J., Helal A., « A programming model for pervasive spaces », *3rd International Conference on Service Oriented Computing*, Amsterdam, The Netherlands, December, 2005.

Kiczales G., Lamping J., Menhdhekar A., Maeda C., Lopes C., Loingtier J.-M., Irwin J., « Aspect-Oriented Programming », *Proceedings European Conference on Object-Oriented Programming*, vol. 1241, Springer-Verlag, Berlin, Heidelberg and New York, p. 220-242, 1997.

Lavirotte S., Lingrand D., Tigli J.-Y., « Définition du contexte : fonctions de coût et méthodes de sélection », *Actes des Secondes Journées Francophones : Mobilité et Ubiquité (UbiMob'05)*, p. 9-12, 2005.

Lyytinen K., Yoo Y., « Issues and Challenges in Ubiquitous Computing », *Communications of the ACM*, vol. 45, n° 12, p. 62-65, 2002.

Maes P., « Concepts and experiments in computational reflection », *OOPSLA87 : Conference proceedings on Object-oriented programming systems, languages and applications*, siacmsn n. 22-12, ACM Press, New York, NY, USA, p. 147–155, 1987.

Mahéo Y., Guidec F., Courtrai L., « Middleware Support for the Deployment of Resource-Aware Parallel Java Components on Heterogeneous Distributed Platforms », *30th Euromicro Conference - Component-Based Software Engineering Track*, p. 144-151, September, 2004.

McKinley P. K., Sadjadi S. M., Kasten E. P., Cheng B., A taxonomy of compositional adaptation, Technical report n° MSU-CSE-04-17, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, 2004.

Medvidovic N., Taylor R., « A classification and comparison framework for software architecture description languages », *IEEE Transactions on Software Engineering*, vol. 26, n° 1, p. 70-93, 2000.

Oprescu J., Service Discovery and Composition in Ambient Networks, Thèse de doctorat, Institut National Polytechnique Grenoble, France, Décembre, 2004.

Pascoe M. J., « Adding generic contextual capabilities to wearable computers », *ISWC '98: 2nd IEEE International Symposium on Wearable Computers*, IEEE Computer Society, Washington, DC, USA, p. 92-99, 1998.

Pauty J., Couderc P., Banâtre M., Synthèse des méthodes de programmation en informatique contextuelle, Rapport de recherche n° 5094, INRIA, Rennes, Janvier, 2004.

Pessemier N., Barais O., Seinturier L., Coupaye T., Duchien L., « A Three Level Framework for Adapting Component-Based Software », *Second International Workshop on Coordination and Adaptation Techniques for Software Entities WCAT05*, July, 2005.

Pessemier N., Seinturier L., Duchien L., Barais O., « Une extension de Fractal pour l'AOP », *Première journée francophone sur le développement de logiciels par aspects JFDLPA 2004*, Paris, France, Septembre, 2004.

Prekop P., Burnett M., « Activities, context and ubiquitous computing », *Computer Communications*, vol. 26, p. 1168-1176, 2003.

Rashid A., Kortuem G., « Adaptation as an aspect in pervasive computing », *Symposium on Principles of Distributed Computing*, Vancouver, Canada, 2004.

Roman G.-C., Picco G. P., Murphy A. L., « Software engineering for mobility: A roadmap », *ICSE'00 : The Future of Software Engineering*, ACM Press, New York, NY, USA, p. 241-258, 2000.

Satyanarayanan M., « Fundamental challenges in mobile computing », *Symposium on Principles of Distributed Computing*, p. 1-7, 1996.

Schilit B., Theimer M., « Disseminating Active Map Information to Mobile Hosts », *IEEE Network*, vol. 8, n° 5, p. 22-32, 1994.

Schmidt D., Stal M., Rohnert H., *Pattern-Oriented Software Architecure - Patterns for Concurrent and Networked Objects*, vol 2, John Wiley & Sons, Ltd, 2000.

Seinturier L., *L'Objet - Développement de logiciels par aspects : JFDLPA 2005*, vol. 12, Lavoisier edn, Hermès, Juin, 2006.

Serafini L., Bouquet P., « Comparing formal theories of context in AI », *Artificial Intelligence*, vol. 155, p. 41-67, 2004.

Springer-Verlag (ed.), *Proceedings of the First International and Interdisciplinary Conference on Modeling and Using Context*, vol. 1688, 1997.

Suvée D., Vanderperren W., Jonckers, V., « FuseJ : An Architectural description language for unifying aspects and components », *Workshop Software-engineering Properties of Languages and Aspect Technologies*, 2005.

Szyperski C., *Component Software, Beyond Object-Oriented Programming*, Addison-Wesley, 1998.

Want R., Hopper A., Falcão V., Gibbons J., « The active badge location system », *ACM Transactions on Information Systems*, vol. 10, n° 1, p. 91-102, 1992.

Weiser M., « The computer for the twenty-first century », *Scientific American Ubicomp Paper*, vol. 1, p. 94-104, September, 1991.

Winograd T., « Architectures for Context », *Human Computer Interaction*, vol. 16, n° 2-4, 2001.