MASTER PRO STIC-ISI (Informatique et Sciences de l'Ingénieur)

RAPPORT DE STAGE DE FIN D'ETUDES 2008

Adaptation Dynamique d'Applications au contexte

Stagiaire : Nicolas FERRY (Master STIC-ISI Filière SAR)

Maître de stage : Stéphane Lavirotte

Tuteur enseignant : Stéphane Lavirotte

Entreprise : Centre National de la Recherche Scientifique
Laboratoire I3S – Équipe Rainbow

3 Mars 2008- 30 Septembre 2008

Résumé :

Les applications logicielles, maintenant distribuées, voire mobiles, nécessitent toujours plus de capacités d'adaptation face à une multitude de contextes d'utilisation (multi-dispositifs, multi-applications, multi-utilisateurs, dans un environnement physique variable). Les travaux existant actuellement proposent des mécanismes permettant de réaliser de telles adaptations. Le problème complémentaire à traiter est l'étude des mécanismes en permettant le déclenchement tout en conservant une approche architecturale innovante de décentralisation particulièrement adaptée au cadre de l'informatique ambiante.

Remerciements

Sans l'aide et l'intervention de nombreuses personnes il m'aurait été difficile de réaliser ce travail, je tiens donc à les remercier tout particulièrement.

Mes remerciements s'adressent en premier lieu à Stéphane Lavirotte qui m'a encadré durant ce stage. Je le remercie pour ces précieux conseils et sa patience. Je remercie également Gaëtan Rey et Jean-Yves Tigli pour avoir été présents et les conseils qu'ils m'ont promulgués tout au long de ce stage. Je les remercie tout les trois pour avoir su se rendre disponible quand j'avais besoin d'eux.

Enfin, je remercie mes parents, qui m'ont toujours laissé libre de mes choix et soutenu quels qu'ils soient.

Table des matières

Chapitre 1

Introduction

1.1 Cadre d'étude

Dans notre vie quotidienne les terminaux mobiles sont de plus en plus accessibles et présents. Ces objets communicants et mobiles nous suivent partout et introduisent une forte variabilité dans les environnements d'exécution des plateformes sensibles au contexte. Cette vision a mené Mark Weiser en 1991 [1] à poser la notion d'**informatique ambiante**, une informatique présente en tout lieu, à tout instant et en toutes choses. Ainsi les environnements ubiquitaires tendent à mettre en jeu de nombreux dispositifs communicants et mobiles et ce de la manière la plus transparente possible. Ces systèmes, pour être utiles et utilisables, doivent faciliter la tâche de l'utilisateur. Il leur est ainsi nécessaire de posséder la capacité de s'adapter à leur environnement, à leur contexte, afin de faire intervenir l'utilisateur le moins possible dans la reconfiguration du système pour qu'il tire au mieux parti de son nouvel environnement

La notion de **contexte** apparaît pour la première fois, associée à l'informatique ambiante, en 1993 dans les publications de Mark Weiser [2] comme l'ensemble des informations à prendre en compte en vue d'une adaptation. En 2001, Anind K Dey propose la définition suivante : « *Le contexte se compose de n'importe qu'elle information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu, où un objet en rapport avec une interaction homme machine, y compris l'utilisateur et l'application.* » [3]. Par la suite dans DEN-ng [4] la définition suivante du contexte est proposée : « *Le contexte d'une entité est une collection de mesures et de connaissances déduites qui décrivent l'état et l'environnement dans lequel une entité existe ou a existé.* ». Ces définitions montrent la difficulté et le manque de maitrise que l'on a de cette notion. Toutefois des progrès se font dans ce domaine et de nouvelles idées viennent enrichir et par conséquent affiner cette définition. Ainsi dans la sienne Dey ne couvre pas les notions de temps et d'état que propose la seconde définition. D'autres idées encore, comme la pertinence des entités en jeux, peuvent entrer en ligne de compte. Nous voyons donc ici que ce domaine est en pleine évolution et que les idées peu à peu sont mieux formalisées.

L'informatique ambiante impose alors certaines contraintes. En effet il convient de considérer dans ce cadre de **multiples** utilisateurs ou encore de multiples dispositifs. Cette multiplicité implique de prendre en compte des entités qui peuvent être **hétérogènes**, qu'il s'agisse de plateformes ou encore de dispositifs. Ces derniers pouvant être **mobiles** tout comme les

utilisateurs il est alors nécessaire de posséder un système **extensible** et capable de prendre en compte les fortes variations qui peuvent intervenir dans son environnement, dans son contexte. Il faut donc prendre en compte cette **dynamicité** de l'environnement et y être **réactif** afin de les considérer le plus rapidement possible pour une bonne utilisabilité. Ainsi le défi que propose l'informatique ambiante est d'être capable de **s'adapter dynamiquement au contexte**.

1.2 Présentation du sujet

On peut noter trois grands moments d'adaptation dans la vie d'une application : lors de son développement, à l'installation ou encore à l'exécution ; c'est ce dernier cas qui nous intéressera plus particulièrement. Le déclenchement d'une adaptation peut être soit décidé par un utilisateur directement ou encore choisi à un instant t donné. Enfin il peut être décidé quand un certain nombre de conditions (qui peuvent être autres que temporels) sont remplies.

Ces conditions sont évaluées à la suite d'une phase d'extraction d'observables à travers différentes entités. Nous pouvons définir un observable comme une variable dont la valeur peut être acquise (à l'aide de capteurs) et ou être calculée par le système (Rey ,2002). D'autre part des informations relatives au contexte peuvent être obtenues à partir de déductions sur ces observables. L'ensemble de ces informations sera ainsi évalué dans l'objectif de déclencher une adaptation. Ainsi les phases de déductions et de calcul de la réaction à obtenir sont à la base des mécanismes de déclenchement d'adaptations tandis que l'évaluation en est le coeur. Dans le cadre de la problématique soulevée dans ce travail consistant en l'étude des conditions de déclenchement d'une adaptation, et par conséquent des mécanismes de prise en compte du contexte, nous présenterons dans un premier temps notre approche architecturale pour y répondre et montrerons son originalité. A plus long terme l'objectif étant d'aider à la construction d'applications avec l'évolution de l'environnement, chaque nouvel objet venant avec ses conditions contextuelles. Permettant ainsi à un utilisateur de construire son application en fonction de ces préférences, de son contexte, là ou aujourd'hui les applications sont développées dans l'optique de répondre à ce qui peut intéresser des utilisateurs en générale et de manière « statique ».

De nombreux intergiciels pour l'informatique ambiante ayant leurs propres caractéristiques existent mais peu d'états de l'art ou de comparaisons de ces derniers apparaissent dans la littérature. Nous réaliserons donc une telle étude dans une seconde partie et présenterons les résultats selon trois axes. A quels besoins de l'informatique ambiante répondent-ils ? Font-ils de l'adaptation et si oui comment ? Prennent-ils en compte le contexte et de quelle manière ?

Dans une troisième partie nous expliquerons et présenterons plus en détails l'intergiciel choisit selon ces critères et nos choix architecturaux pour finalement proposer une possible mise en oeuvre de ces mécanismes de déclenchement d'adaptations.

Chapitre 2

Architecture logicielle : une approche originale

Dans les systèmes informatiques différentes approches d'architectures logicielles sont envisageables. Chacune d'entre elles ayant ses intérêts et étant plus ou moins bien adaptée à différentes situations. On peut distinguer trois grandes familles d'architectures que sont les systèmes : **centralisés** et distribués qui regroupe les **partiellement décentralisés** et **décentralisés**. Ces derniers peuvent être classés comme suit :

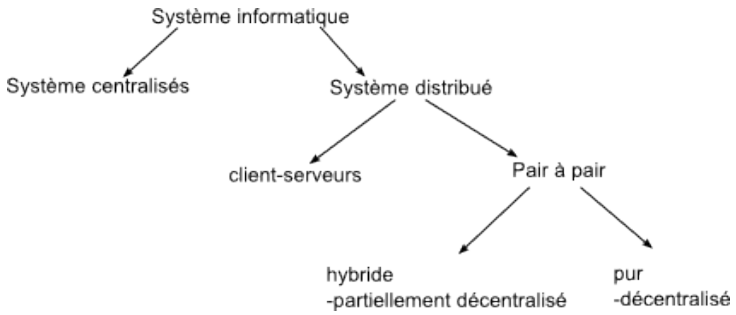

FIG. 2.1 – Classement des systèmes informatiques

Dans les architectures centralisées une entité centrale gère les autres éléments et informations de l'infrastructure. Ce qui en fait un goulot d'étranglement et un point sensible du système. Dans les systèmes partiellement décentralisés certaines composantes du réseau jouent un rôle plus important que les autres services, celles-ci gèrent en générale les mécanismes de découverte ainsi que ceux de mise en relation des différents éléments de l'infrastructure. Ces serveurs ont une fonction d'annuaire. Enfin les systèmes totalement décentralisés forment un réseau non structuré dans lequel chaque élément tient le même rôle.

Ces derniers permettent d'obtenir un partage et une réduction des coûts entre les différents pairs tout en conservant une bonne robustesse grâce à l'absence d'entité centrale au rôle prépondérant. En outre ces architectures offrent la meilleure extensibilité, la meilleure garantie de passage à l'échelle. La puissance dans ces systèmes s'obtient grâce à l'agrégation des ressources disponibles. Toutes ces caractéristiques concourent à l'accroissement de l'autonomie du système.

Nous voyons donc que ces différents points collent aux besoins de l'informatique ambiante. En effet Mark Weiser[2] parlait bien en 1991 d'une informatique présente en tout lieu et à tout instant grâce à une multiplicité de petit dispositifs intelligents et mobiles dont l'accumulation fournirait une puissance de calcul conséquente. Cette multiplicité de dispositifs en jeu et leur mobilité requiert des systèmes capables de supporter des architectures de taille et d'être extensibles. De plus cette multitude de dispositifs, d'entités, implique de prendre en compte de nombreuses informations et dans le cadre de la sensibilité au contexte de conserver une représentation du contexte qui pour une architecure centralisée revient à modéliser le monde entier. On retrouve également dans les définitions de l'informatique ambiante et des systèmes décentralisés la notion d'agrégation de la puissance des différentes entités utilisées mais aussi de système à topologie fortement variable. Enfin Weiser évoque de « petits » dispositifs qui, pris individuellement, n'offrent que peu de ressources, ce qui nécessite de partager les coûts entre les différentes entités comme le propose les systèmes distribués en général. Tous ces points nous ont donc incités à nous orienter vers une approche architecturale originale de prise en compte du contexte décentralisée. Nous verrons plus loin les conséquences impliquées par ce choix.

De nombreux intergiciels pour l'informatique ambiante et sensibles au contexte existent. Mais il apparait que ces plateformes applicatives du domaine ne permettent pas d'avoir une approche décentralisée répartie. Une courte présentation de l'ensemble des intergiciels étudiés est proposée en Annexe (cf. Annexe 1). D'autres plateformes dites de services communs [14] ne sont liées à aucun domaine et par conséquent à priori pas à celui de la prise en compte du contexte. Toutefois elles peuvent être étendues dans cette optique. Parmi ces plateformes nous considérerons celles d'entre elles qui nous offriront la possibilité de prendre une orientation décentralisée. On retrouve trois granularités de centralisation des informations pour la prise en compte du contexte.

- Centralisée au niveau du middleware
- Centralisée pour une application
- Centralisée à un composant

Middleware	**type**	**Type d'architecture**				**Mise en oeuvre**
		Centralisé middleware	Centralisé application	Centralisé composant	Décentralisé	
Gaia	applicative	X				Dans l'entité
RCSM	applicative			X		Une interface contenant les règles par composant
CARMEN	applicative	X				Dans un LDAP spécialisé
Amigo	applicative	X				Dans l'entité ANS
CORTEX	services communs				X	Règles associés aux objets sensibles
Aura	applicative	X				Dans l'entité du middleware environnement manager
CAMidO	applicative		X			Interfaces associées à une application
CARISMA	applicative		X			Métadonnées associées à une application
Oxygen	applicative	X				Dans l'entité du middleware core
SAFRAN	applicative			X		Des scripts sont associés aux composants dynamiquement
WComp	services communs				X	Aspects d'assemblages

Seules les plateformes de services communs nous permettent potentiellement d'obtenir une approche décentralisée de la prise en compte du contexte. Les autres plateformes peuvent difficilement évoluer vers des architectures décentralisées. Mais avant de porter notre choix sur un intergiciels en particulier nous les étudierons plus en détails afin de vérifier qu'ils sont biens adaptés à l'informatique ambiante puis s'ils sont adaptatifs. Enfin nous nous assurerons qu'ils font bien de la prise en compte du contexte. D'autre part nous pourrons de cette manière identifier les services manquant aux plateformes de services communs pour s'orienter vers le domaine de la prise en compte du contexte.

Chapitre 3

Etude des intergiciels existants

Un intergiciel est une couche applicative servant d'intermédiaire entre plusieurs applications, mais aussi entre applications et systèmes, et ce généralement de manière distribuée. Cette couche offre des services de haut niveau principalement liés aux communications.

De nombreuses recherches en informatique ambiante portent sur la mise en place d'intergiciels adaptatifs et sensibles au contexte qui peuvent avoir des motivations diverses. Si Aura [6] et Oxygen [7] ont pour objectif le déplacement de la tâche utilisateur dans un environnement ubiquitaire, Amigo [8] s'adresse plus particulièrement à la découverte ainsi qu'à la gestion de nouvelles entités, de nouveaux dispositifs. S'ils sont nombreux et pour la plupart bien documentés, peu de travaux les étudient ou en proposent un état de l'art. Dans cette section nous nous pencherons donc sur ces middleware afin de nous conforter ou non dans notre choix d'une plateforme de services communs.

3.1 Des intergiciels pour l'informatique ambiante

Dans un premier temps il convient de vérifier que ces intergiciels s'appliquent bien aux problématiques de l'informatique ambiante. Nous identifierons donc ici les défis de l'informatique ambiante puis vérifierons si les caractéristiques des différentes plateformes permettent d'y répondre.

3.1.1 Caractéristiques des intergiciels et besoins de l'informatique ambiante

Dans le cadre de l'informatique ambiante et de la prise en compte du contexte les intergiciels doivent répondre au mieux à certains critères. Ainsi la multiplicité et l'hétérogénéité des entités mises en jeu dans un environnement ubiquitaire doit être prise en compte.

- **Hétérogénéité** : Capacité à prendre en compte différents langage, OS, dispositifs

De même la haute variabilité de l'environnement et la mobilité des entités le composant nécessitent un système devant s'adapter, réagir au plus vite aux changements.

- **Réactivité** : Capacité à réagir le plus rapidement possible aux changements de contexte par exemple avec l'utilisation d'événements ou de mécanismes de publish/subscribe
- **Mobilité** : Capacité d'interagir avec le système dans des lieux distincts et de prendre en compte des entités mobile

Si traditionnellement dans les applications classiques les entités en jeux sont connues ainsi que leur environnement, les besoins de l'informatique ambiante que nous avons évoqués précédemment montrent que cette vision statique d'une application est trop restrictive. Il est important pour les systèmes d'avoir la possibilité de découvrir dynamiquement les services présents dans leur environnement.

- **Découverte** : Capacité à découvrir dynamiquement les entités se trouvant dans l'environnement

Ainsi en étudiant les différents middlewares présentés dans la section précédente nous observons qu'ils présentent les caractéristiques suivantes :

Middleware	**Hétérogénéité**	**Réactivité**	**Mobilité**	**Découverte**
Gaia		X	X	X
RCSM		X	X	
CARMEN		X	X	
Amigo	X		X	X
Aura		X	X	X
CAMidO			X	
CARISMA		X	X	
Oxygen			X	X
SAFRAN		X		
CORTEX	X	X	X	X
WComp	X	X	X	X

Souvent ces systèmes s'attachent plus particulièrement à prendre en compte certaines caractéristiques ; Amigo [9] se concentre d'abord sur l'hétérogénéité et la découverte de nouveaux dispositifs grâce à des technologies comme SLP ou UPnP. Tandis que CORTEX [10] adresse plus particulièrement le problème de la réactivité, ainsi une application dans CORTEX est un assemblage d'objets sensibles communiquant à l'aide d'événements, chaque objet possédant un mécanisme de prise en compte du contexte. SAFRAN [11] met en avant les avantages de la programmation par aspects et de la séparation des préoccupations qu'elle permet pour réaliser une adaptation. Gaia [12] et Aura [6] s'attachent plus particulièrement à prendre en compte le problème de la migration de la tâche utilisateur et par conséquent traite le problème de la mobilité en priorité. Enfin Oxygen [7] s'intéresse aux interactions homme machines grâce à des techniques de reconnaissance vocale et de traitement d'images. Nous voyons donc qu'il existe de nombreux axes de recherches dans le cadre des intergiciels pour l'informatique ambiante. Il apparait enfin que les plateformes de services communs CORTEX et WComp sont les plus complètes et permettent de répondre à ces quatre défis de l'informatique ambiante. Elles peuvent donc servir de base pour une orientation vers un domaine plus particulier de l'informatique ambiante. Afin de mieux comprendre de quelle manière il est possible de répondre à ces problèmes nous étudierons dans la section suivante les paradigmes utilisés.

3.1.2 Paradigmes mis en oeuvres

Pour répondre à toutes ces préoccupations les intergiciels peuvent se baser sur différents paradigmes, offrant chacun des caractéristiques bien précises. Ils permettent par conséquent de satisfaire différents besoins de l'informatique ambiante. Ainsi l'utilisation de la programmation événementielle permet d'avoir un couplage faible entre les différentes composantes d'un système, tout en offrant à la manière d'un mécanisme de publish/subscribe un bon support pour une réactivité et une dynamicité maximale. De leurs cotés les architectures orientés services apportent des solutions dans la gestion de l'hétérogénéité des entités en jeu. Mais aussi des réponses aux problèmes d'interopérabilité que l'on peut trouver dans des environnements aussi variés que ceux de l'informatique ambiante, toujours en gardant une approche distribuée. L'utilisation de la programmation par composant qui implique une approche « boite noire » des entités logicielles autorise une forte réutilisabilité des composants. Ceci à entre autre pour effet de faciliter les opérations de maintenance d'un système. Couplée à ces différentes approches la programmation par aspects facilite également la maintenance d'un système et permet surtout de jouir d'une modularité transverse, c'est à dire d'une séparation des services d'une application et de ses propriétés non fonctionnelles (sécurité, persistance ...) [13].
En effet, avec la programmation par aspects, les services d'une application et ses propriétés non fonctionnelles correspondent à des aspects découplés les uns des autres. Une application est alors obtenue par enchevêtrement, tissage des différents aspects. Cette composition peut être obtenue dynamiquement grâce au tisseur d'aspects. Tous ces paradigmes peuvent être combinés afin de tirer au mieux parti de leurs avantages Chaque intergiciel utilise donc les

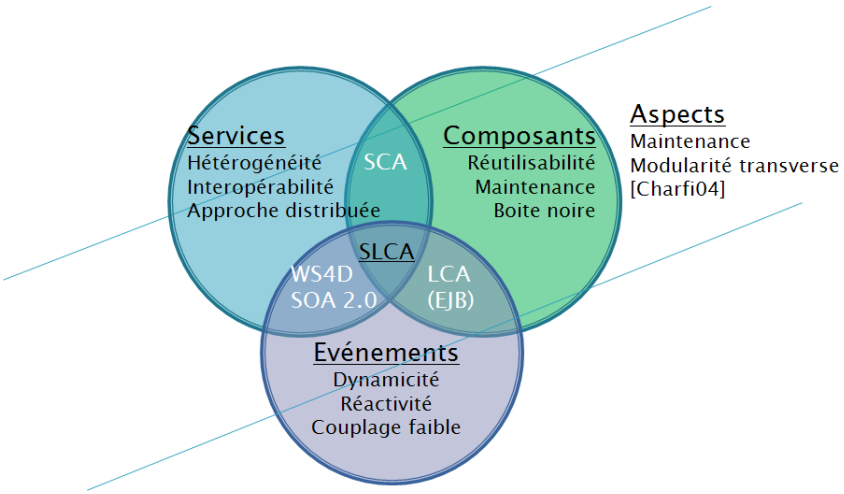

FIG. 3.1 – Approches multi-paradigmes

paradigmes les plus appropriés à la résolution des problèmes qu'il adresse. On remarquera toutefois l'importance de la réactivité du système aux changements dans son environnement qui pousse la plupart des intergiciels à utiliser la programmation événementielle. De plus il apparait que la programmation par aspects reste encore une approche marginale.

	Composants	Objets	Services	Evénements	Aspects
Gaia	X				
RCSM		X		X	
CARMEN		X		X	
Amigo		X	X	X	
Aura			X	X	
CAMidO	X			X	
CARISMA			X		
Oxygen		X			
SAFRAN	X			X	X
CORTEX		X		X	
WComp	X		X	X	X

Parmi les défis que les intergiciels pour l'informatique ambiante doivent relever nous avons vu que la capacité d'un système à s'adapter à son contexte est primordiale.

En conclusion de cette section de l'étude il apparait que les plateformes de services communs CORTEX et WComp sont les plus complètes, puisque non destiné à un domaine en particulier,et permettent de répondre aux quatre défis de l'informatique ambiante mis en avant dans cette section. Les autres plateformes apportent plus précisement une réponse à quelques défis de l'informatique ambiante liés à leur domaine.

3.2 Des intergiciels adaptatifs

Avant de sélectionner un intergiciel il est donc nécessaire de vérifier qu'ils proposent des mécanismes d'adaptation. D'autre part afin de bien cerner les besoins des mécanismes de déclenchement d'adaptation il est important de l'étudier préalablement.

3.2.1 L'adaptation

Adapter : *« ajuster une chose à une autre »- Littré*

Comme nous l'avons évoqué précédemment dans le cadre de l'informatique ambiante et avec la haute variabilité de l'environnement qu'il implique, il est important que les applications s'adaptent à ce qui les entoure. L'adaptation logicielle peut prendre plusieurs formes, ainsi on parlera de système adaptable lorsqu'un utilisateur peut interagir avec le système et par ce biais le modifier, le personnaliser. Un système adaptatif identifie une situation et s'y adapte ; le déclenchement de cette modification du système peut avoir pour origine une intervention humaine ou encore un certain nombre d'observations. Lorsque l'intervention humaine n'est pas requise dans le cadre de l'adaptation d'un système on parle alors de systèmes auto-adaptatifs. Dans le cadre de cette étude nous nous intéresserons aux systèmes adaptatifs ou auto-adaptatifs.

On distingue deux techniques permettant de réaliser de l'adaptation dynamique logicielle : l'adaptation paramétrée et la reconfiguration [14]. 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. A l'inverse la reconfiguration vise à échanger des algorithmes, stratégie d'une application par d'autres dans l'optique de coller au mieux à une situation donnée.

Lorsque l'on cherche à adapter un système trois questions se posent : Quoi adapter ? Comment adapter ? Quand adapter ? Différentes techniques, « *le comment* », peuvent être utilisées pour réaliser une adaptation parmi lesquels :

- **MOP (Meta Object Protocol) :** Grâce à l'utilisation de langage supportant la réflexion il est alors possible de modifier le comportement de l'application [14]. La réflexion se découpe en deux sous-mécanismes : l'introspection qui permet à un système d'analyser son propre état et l'intercession qui permet au système d'agir sur lui-même et par conséquent de modifier son comportement.
- **Proxy :** Des proxies sont utilisés à la place des objets pour les représenter et peuvent ainsi rediriger des appels de méthodes vers différentes instances [14].
- **Container :** Le conteneur d'un composant gère ses propriétés non fonctionnelles et contrôle les interactions du composant avec l'extérieur. Il peut également gérer ses composants internes. Il peut donc être réutilisé dans le but de paramétrer ou de reconfigurer un assemblage de composants pour modifier le comportement d'une application.
- **Tissage :** La programmation par aspects offre la possibilité de créer des systèmes modulaires en permettant de séparer les préoccupations et plus particulièrement les parties fonctionnelles des non-fonctionnelles. Ainsi les aspects peuvent être des comportements qui se tissent avec l'application existante. Cet entrelacement des aspects, le tissage, est réalisé par un tisseur. Ce dernier permet d'ajouter ou de supprimer dynamiquement à l'exécution des fragments de code relatifs à des préoccupations transverse [14].
- **Interception middleware :** Les appels de méthode et le passage de réponses à travers les couches d'un intergiciel peuvent être interceptés et redirigés vers différentes entités liées au middleware [14].
- **Génération de code :** Du code est généré (par exemple par compilation) puis inséré dans l'application. Il devient par exemple possible de compiler du code alors que l'intergiciel est en cours d'utilisation. Cette technique reste difficile à mettre en place et à un coût en terme de performances qui peut être néfaste à la dynamicité du système.

La capacité d'un système à s'adapter 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. D'autre part ces adaptations peuvent être dynamiques ou statiques, soit respectivement se dérouler à l'exécution ou lors de la phase de chargement ou de compilation d'une application. Dans le cadre de l'informatique ubiquitaire des adaptations dynamiques sont souhaitables.

Enfin l'adaptation d'un système à son environnement peut être soit réactive, dans une situation donnée on réalise une action d'adaptation, soit proactive, dans une situation donnée on prépare de nouvelles actions pour les appels à venir. Il peut s'agir par exemple de rediriger des appels futurs à certaines méthodes lorsque l'on a identifié une situation donnée.

Ainsi ces intergiciels étant tous adaptatifs nous étudierons dans la section suivante les mécanismes utilisés pour y parvenir. Ce qui nous permettra de mettre en avant les intergiciels ayant les approches les plus performantes.

3.2.2 Mécanismes d'adaptation

Comme nous l'avons vu précédemment, de nombreuses techniques existent pour effectuer une adaptation. Le choix de ces techniques dépend en grande partie des paradigmes de programmation utilisés par les intergiciels, le tissage ne peut avoir lieu que si l'on fait des aspects et l'utilisation de container que si des composants sont utilisés. Plusieurs d'entre elles peuvent être utilisées par un même système, en effet le type d'adaptation visé peut être différent selon la technique utilisée. Ainsi l'interception par le middleware peut être utilisée pour réaliser des adaptations proactives (les prochains appels aux méthodes seront redirigés vers d'autres entités) tandis que le tissage reste plus adapté aux réactives.

	MOP	**Proxy**	**Container**	**Tissage**	**Interception middleware**	**génération**
Gaia			X			
RCSM			X			X
CARMEN		X				
Amigo						
Aura		X				X
CAMidO			X		X	X
CARISMA	X					
Oxygen	X					
SAFRAN			X	X		
CORTEX	X					
WComp			X	X		X

Certaines techniques permettent d'obtenir une bonne séparation des préoccupations fonctionnelles d'une application et de l'adaptation. Ainsi le tissage et l'utilisation des aspects apportent à SAFRAN et WComp ce résultat tout comme la génération pour CAMidO et RCSM. D'autre part l'utilisation de métadonnées peut aboutir au même résultat. Ainsi tout ces middleware sont adaptatifs mais ces derniers semblent les plus intéressant, toutefois ces caractéristiques ne sont pas suffisantes pour décider quelle plateforme choisir.

3.2.3 Adaptation réactive et proactive

Comme nous l'avons évoqué auparavant une adaptation réactive est déclenchée lors de la découverte d'un contexte pertinent ; à l'inverse une adaptation proactive prépare de nouvelles actions pour les appels à venir lors de la détection d'un contexte pertinent. Cette dernière approche peut être réalisée grâce aux techniques d'interception par le middleware ou à l'aide des containers qui peuvent rediriger des appels de méthodes. CARISMA [15] et CAMidO [16] à l'aide de ces processus réalisent ce type d'adaptation. D'autres approches plus complexes permettent d'obtenir un autre type d'adaptation proactive, il s'agit alors d'anticiper la détection d'un contexte pertinent grâce à un historique des contextes observés ou par un mécanisme d'apprentissage comme le fait Amigo [8]. Toutefois il est important de noter que peu d'intergiciels réalisent des adaptations proactives.

	Adaptation réactive	Adaptation proactive
Gaia	X	
RCSM	X	
CARMEN	X	
Amigo	X	X
Aura		X
CAMidO	X	X
CARISMA	X	X
Oxygen	X	
SAFRAN	X	
CORTEX	X	
WComp	X	

En conclusion de cette section de l'étude il apparait que les intergiciels WComp, CAMidO, RCSM et SAFRAN utilisent les mécanismes d'adaptation les plus intéressant puisqu'ils offrent une bonne séparation des préoccupations. D'autre part les intergiciels Amigo, CAMidO et CARISMA permettent d'utiliser à la fois des adaptations réactives et proactives ce qui les rend également particulièrement attrayant.

3.3 Des intergiciels sensibles au contexte

Après avoir étudié la capacité d'adaptation des intergiciels et les réponses qu'ils apportent à la problématique de l'informatique ambiante, nous nous intéresserons dans ce chapitre à leur sensibilité au contexte. Les plateformes applicatives étudiés prennent toutes plus ou moins en compte le contexte. Les plateformes de services communs quand à elles ne sont à priori pas liées à ce domaine. Nous nous pencherons donc dans un premier temps sur la notion de contexte puis nous évaluerons les capacités de chacun d'entre eux à le considérer et leur bon respect du cycle de l'adaptation au contexte que nous avons défini. Ce qui nous permettra par la suite d'entrevoir ce qui peut manquer aux intergiciels CORTEX et WComp dans ce domaine.

3.3.1 Le contexte

Etymologiquement, le mot contexte qui a pour origine co-text est définit comme l'ensemble du texte qui entoure un extrait et qui éclaire son sens. Avec le temps sa signification a évolué pour se définir comme suit :

Contexte : *« Ensemble des circonstances dans lesquelles se produit un événement, se situe une action »-Larousse*

Le contexte est une notion fondamentale en psychologie, il est défini comme un ensemble d'éléments d'une situation y compris l'objet de référence auquel il est rattaché. Ce concept est utilisé aussi bien dans les études sur la mémoire, par exemple quelles sont les conditions qui font que l'on se souvient de certaines choses plus que d'autres, que sur le langage.
En informatique prendre en compte le contexte permettrait d'obtenir de nombreuses avancés. Ainsi nous pouvons imaginer des applications qui s'adapteraient à leur environnement d'exécution (j'adapte la luminosité de mon écran aux applications utilisées et à l'autonomie de ma

batterie) ou encore à leur environnement global (la luminosité est fonction des applications que j'utilise et de ma batterie mais aussi de la lumière du jour et du nombre de personne autour de la machine). On peut alors imaginer des systèmes ou les interactions sont facilitées par la prise en compte de ce qui les entoure. De nombreuses définitions du contexte ont étés posées dans cet objectif. Dey le défini comme : « *Le contexte se compose de n'importe qu'elle information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu, où un objet en rapport avec une interaction homme machine, y compris l'utilisateur et l'application.* » [3]. Pascoe dans [17] parle du contexte comme n sous-ensembles d'états physiques et conceptuels ayant un intérêt pour une entité particulière et fait apparaitre la notion de pertinence associé au contexte. Chen et Kotz [18] pose 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 alors apparaitre la notion d'informations passives qui décrivent le comportement à avoir et les informations actives qui déclenchent ce changement de comportement. Ces notions peuvent être ramenées à celles d'observables et de préférences. Les premiers étant plutôt actifs tandis que les secondes sont passives. Il est possible de définir ces derniers comme suit :

- **Observables** : données obtenues par l'observation de l'environnement à l'aide de dispositifs particuliers appelés « *observers* »
- **Préférences** : données fournies par un utilisateur caractérisant une application, une entité, représentant les souhaits de l'utilisateur.

Nous voyons également qu'il est possible de créer différentes catégories de contexte et par conséquent de conditions susceptibles d'être à l'origine d'une adaptation. Ainsi une adaptation peut être déclenchée volontairement par une interaction d'un utilisateur avec le système, on parle alors de contexte explicite. A l'inverse elle peut être déclenchée par un changement dans l'environnement, par un changement de situation. On s'intéresse alors au contexte physique fourni par des capteurs ou au contexte logique obtenu par du monitoring ou l'utilisateur. Pour tenir compte de ces contextes physiques et logiques il faut donc s'intéresser aux :

- **Contexte utilisateur** qui concerne toutes les informations qui peuvent être relatives à un utilisateur, sa localisation, son humeur, la tâche qu'il est en train de réaliser
- **Contexte temporel** qui traite de tout ce qui est relatifs aux instants, date, historique
- **Contexte machine** qui concerne l'environnement d'exécution, ressources disponibles, monitoring
- **Contexte environnemental** qui est relatif à tout ce qui entoure le système, luminosité, bruit, température

Cette catégorisation nous permettra par la suite de définir des grandes familles de conditions pouvant déclencher une adaptation.
De nombreuses recherches portent aujourd'hui sur la modélisation du contexte, et proposent des approches variées :

- Paire clé valeur : Il s'agit de la structure de données la plus simple et la plus facile à gérer. A une clé est associée une valeur.
- Schema XML : L'utilisation du XML permet d'obtenir une représentation hiérarchique

des données contextuelles. Ces structures peuvent être vérifiées, parcourues et distribuées simplement. De plus le langage pouvant être relativement verbeux son niveau de formalisme fait qu'il reste abordable par les utilisateurs.

- UML et modèles graphique : UML reste l'approche la plus utilisées pour réaliser des modèles. Ce genre de représentation très accessible offre une grande souplesse dans la modélisation des données.
- Modèles logiques : Le contexte est exprimé comme un ensemble de faits ou de règles. Le principal inconvénient de ce type de modélisation reste son haut niveau de formalisme.
- Ontologies : Cette approche semble être la plus prometteuse, elle permet de poser des concepts et de spécifier des relations entre différents éléments composant l'ontologie. D'autre part cette méthode permet d'ajouter de la sémantique aux éléments contextuels.

La modélisation du contexte est une étape importante pour sa prise en compte. Une fois modélisées les données pourront être analysées en vue de déclencher une adaptation.

3.3.2 La prise en compte du contexte

La prise en compte du contexte par les applications repose sur différents mécanismes, en général le modèle présenté sous la forme de quatre couches que sont la capture d'observables, leur transformation en observables symbolique, l'identification de la situation et l'exploitation [19] s'applique bien. La plupart des intergiciels présentés respectent ce cycle. Cette étude nous permettra de mettre en avant les phases et services nécessaires à une bonne prise en compte du contexte.

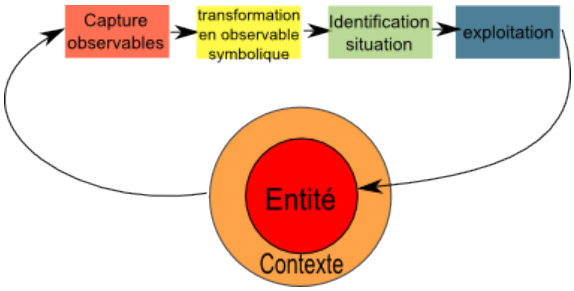

FIG. 3.2 – Cycle de la prise en compte du contexte

Capture et transformation d'observables

L'acquisition d'informations relatives au contexte s'effectue à l'aide de sondes, de capteurs. Traditionnellement, lors de la réalisation de systèmes utilisant de tels mécanismes, les développeurs choisissent leurs sondes statiquement. Malheureusement la haute variabilité nécessaire à un environnement ubiquitaire rend cette solution de moins en moins acceptable. Ainsi des mécanismes de découverte dynamique de services ont étés mis en place grâce à des protocoles comme UPnP, SLP, DPWS ou Jini. Cependant ces mécanismes peuvent subir quelques limitations. Par exemple l'intergiciel Gaia [20] ne permet de découvrir que des entités déjà connues. Oxygen [7] peut détecter uniquement une catégorie particulière de dispositifs créés pour l'occasion : les H21. CARISMA [15] ajoute la notion de qualité de service à sa découverte mais ne fonctionne en conséquence qu'avec des ressources locales. Enfin CARMEN [21]

ne peut découvrir que quelques ressources disponibles à travers l'internet. Toutefois quelques autres intergiciels font de la découverte dynamique autonome comme CORTEX [22], Amigo [9] ou encore Aura [23].

La découverte dynamique autonome de nouveaux services, qui comme nous l'avons vu parfois reste optionnelle, permet de définir l'infrastructure capable de fournir des informations relatives au contexte. Il apparait que ces seules données ne sont parfois pas suffisantes pour avoir un raisonnement opportun. Ainsi l'étape de transformation en observables symboliques peut incorporer des mécanismes d'interprétation de ces informations dans l'optique d'en déduire des données de plus haut niveau. Différentes techniques peuvent être utilisées dans cette optique :

- Ontologies
- Systèmes de règles
- Réseaux bayésiens
- Graphes d'opérateurs [?]

Une fois ces informations acquises et modélisées dans un des formats présentés auparavant, celles-ci doivent être analysées, évaluées dans l'optique d'adapter le système à son environnement au mieux. Pour plus de lisibilité et d'utilisabilité les données doivent être filtrées pour, par exemple, ne pas prendre en compte les trop faibles variations détectées par un capteur.

Identification d'une situation

Une fois collectées, les données relatives au contexte doivent-être évaluées afin de calculer la réaction au changement dans le contexte à obtenir (dans notre cas une adaptation). Comme évoqué précédemment l'évaluation doit donc prendre en compte les différentes familles de contexte que ce soit le contexte explicite, le contexte logique et le contexte physique. Ces derniers nous poussant alors à nous intéresser aux catégories de contexte physique et logique. Tout cela à travers différents mécanismes comme les ontologies, les règles ECA, ou encore la logique floue. Les mécanismes d'évaluation doivent disposer d'interfaces 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. Les contextes observés faisant parti de la situation. Ainsi posséder un tel historique permet de coller à la définition du contexte donnée précédemment, à savoir : « le contexte d'une entité est une collection de mesures et de connaissances déduites qui décrivent l'état de l'environnement dans lequel une entité existe *ou a existé* ». De plus, l'étude des observables doit être couplée à celle des différentes préférences fournies par un utilisateur et qui sont un complément d'informations. Enfin l'utilisation de techniques d'apprentissage peut être également envisagée, en effet 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.

Evaluation des intergiciels

Ainsi les middlewares CARMEN [21], RCSM [24], AMIGO, CARISMA [15], CAMidO [16], Gaia [12] ou encore SAFRAN [11] collent à ce cycle. Toutefois AMIGO [25] apporte une modification intéressante puisqu'il ajoute après sa modélisation du contexte et avant

l'évaluation une phase d'apprentissage tandis qu'Aura la place après. Il est important de noter que dans certains cas la phase « transformation en observables symboliques » peut soit être une simple modélisation ou prendre en compte également des déductions d'informations relatives au contexte de plus haut niveau. SOCAM [26] réalise également cette étape d'apprentissage mais ne réalise pas d'adaptation (que les applications doivent réaliser). Enfin Oxygen [7] utilise une approche beaucoup plus centrée utilisateur et leurs propose de nombreuses interfaces hommes-machines. Nous avons donc vu apparaitre de nouvelles étapes : l'apprentissage ou encore l'intervention de l'utilisateur. D'autre part plus que la simple utilisation d'informations collectées relatives au contexte la plupart des intergiciels prennent également en compte des préférences (utilisateurs, dispositifs, applications). Celles-ci peuvent offrir la possibilité de décrire quand déclencher ces adaptations mais aussi contenir de nombreuses informations comme la portée (public/privée) ou encore concernant la sécurité...

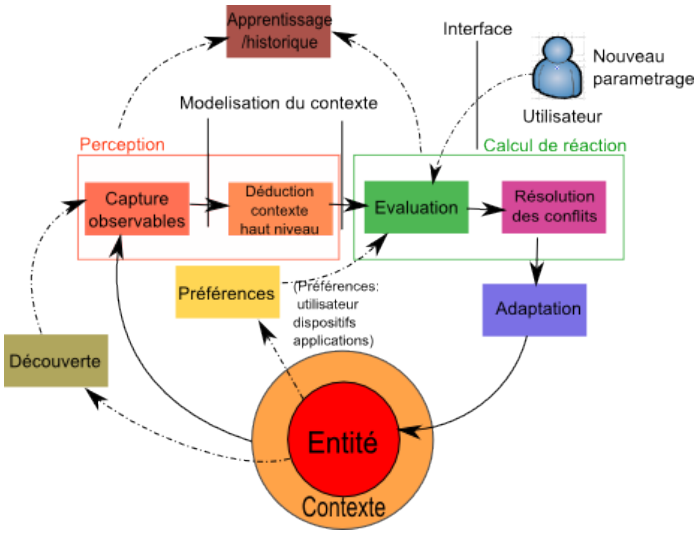

FIG. 3.3 – Cycle détaillé de la prise en compte du contexte

A partir de ces constats nous avons pu construire le cycle précédent dans lequel nous retrouvons bien les éléments de base que sont l'acquisition et la transformation des données relatives au contexte, le calcul de la réaction et l'adaptation en découlant. Toutefois nous voyons donc que des précisions doivent être prise en compte dans l'optique de définir ce qu'est le cycle de l'adaptation au contexte. Nous évaluerons donc les différents intergiciels à partir de ce dernier afin de faire apparaître plus précisément les faiblesses et avantages des plateformes.

	Découverte dynamique de nouvelles entités	**Déduction d'infos de plus haut niveau**	**Historique des contextes observés**	**Apprenti-ssage**	**Utilisation de préférences**	**Interface utilisateur**
Gaia	x	X	X			X
RCSM	x				X	
CARMEN	x				X	X
SOCAM		X	X			X
Amigo	X	X		X	X	X
Aura	X	X	X	X	X	X
CAMidO		X	X		X	
CARISMA	x					X
Oxygen	x	X			X	X
SAFRAN					X	
CORTEX	X					
WComp	X					X

En conclusion, il apparait que les intergiciels Aura et Amigo sont les plus complets et permettent de répondre aux mieux à la problématique de la prise en compte du contexte. Les plateformes de services communs WComp et CORTEX sont à l'inverse que peu prévues pour cela. Mais nous avons désormais identifiées les étapes nécessaire à une bonne prise en compte du contexte.

Chapitre 4

Notre choix : WComp

Comme nous l'avons vu dans cette étude, la plateforme WComp [27] permet de répondre correctement à la plupart de nos attentes. Nous avons vu que cette plateforme de services communs nous offrait la possibilité de travailler avec une architecture décentralisé ainsi que sa capacité à répondre aux besoins de l'informatique ambiante. D'autre part cet intergiciel propose un mécanisme d'adaptation performant, cependant il ne prend que très partiellement en compte le contexte dans lequel il se trouve. En effet la plupart des mécanismes de prise en compte du contexte ne sont pas encore présents dans WComp, seule la découverte dynamique autonome de nouveaux services ainsi qu'une interface graphique de configuration pour des mécanismes d'adaptations existent. Les adaptations sélectionnées par un utilisateur peuvent s'appliquer si l'infrastructure le permet. Le choix d'un autre middleware aurait impliqué de trop grandes modifications de son architecture et par conséquent de la plupart des mécanismes que nous avons étudiés précédemment pour obtenir de la décentralisation. Avec le choix de WComp, seule la prise en compte du contexte est à amélioré en fonction des besoins identifiés précédemment. Dans les sections suivantes nous présenterons plus en détail cet intergiciel.

4.1 WComp, SLCA

WComp, développé par l'équipe Rainbow, est une plateforme de composition de services par assemblage de composants. Il s'agit d'une implémentation du modèle SLCA (Service Lightweight Component Architecture) qui est un modèle multi-paradigmes utilisant : services, composants légers et événements. Dans ce middleware une application prend la forme d'un ensemble de composants légers et de services communiquant entre eux à l'aide d'événements. L'architecture de WComp s'organise autour de container et de designers. Les **containers** gèrent à l'exécution un assemblage de composants légers ; ils représentent un service composite et exposent une interface UPnP. Les **designers** permettent de manipuler les applications, via les containers. Nous pouvons citer comme exemple de designer l'UPnPProxyWizard qui est capable de générer et d'instancier les proxies de services UPnP. Cette technologies à été choisie pour implémenter des web services pour dispositifs, car elle couple à la fois des mécanismes de découverte de nouveaux services et des communications événementielle.

UPnP est une implémentation des services web pour dispositifs, il s'agit d'un standard de Microsoft et Intel. UPnP est une amélioration de Plug'n'play pour que ce dernier s'étende aux réseaux locaux. On y trouve trois types d'entités : [28]

- Les « devices » qui sont des entités se composant de services et d'autres devices
- Les « services » qui sont les plus petites entités pouvant être définies proposent une action
- Les « control point » qui permettent de découvrir et de contrôler des devices

Les fonctionnalités qu'UPnP propose sont les suivantes :

- **Adressing :** acquisition d'une adresse IP dans le réseau
- **Discovery :** mise en relation des entités du réseau avec SSDP. Cette phase est utilisée par les control points ainsi que les device pour s'identifier sur un réseau puis pour confirmer leurs présences.
- **Description :** phase visant à fournir aux points de contrôles des informations décrivant les devices, services. Cette description est écrite en XML.
- **Control :** permet au point de contrôle d'envoyer des commandes aux devices
- **Eventing :** informer les points de contrôle des changements d'état.
- **Presentation :** un device peut fournir une URL de présentation des devices.

Un assemblage de composants se trouve donc dans un container, un service composite. Un service composite peut contenir des représentants vers d'autres services composites ou services. Ainsi une application peut prendre la forme d'une architecture de services composites disséminés, répartis sur différentes entités. Un service composite exporte deux interfaces : une première dite fonctionnelle qui permet de communiquer avec les fonctionnalités proposées par le service, une seconde 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 [29].

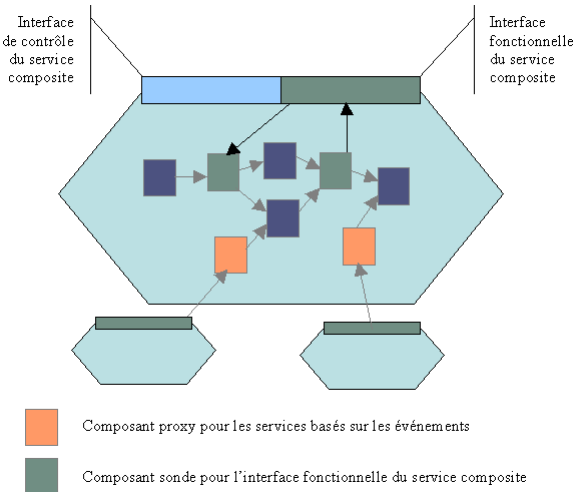

FIG. 4.1 – Service Composite

Tous ces points permettent à SLCA de répondre aux problèmes, évoqués précédemment, qui se posent dans le cadre de l'informatique ambiante, à savoir l'hétérogénéité des entités en jeux grâce à l'utilisation des services, mais aussi aux problèmes de la mobilité ou encore de la découverte dynamique grâce aux WebServices pour dispositifs. Les événements quand à eux apportent la réactivité nécessaire dans de tels environnements (cf table x).

Les mécanismes d'adaptation nécessaires à la prise en compte du contexte sont pour leur part implémentés grâce aux aspects d'assemblages. Ces derniers utilisent les possibilités offertes par l'interface structurelle des services composites puisqu'elles permettent de modifier un assemblage.

4.2 Les aspects d'assemblages

Comme leur nom l'indique les aspects d'assemblages (AAs) sont inspirés de la programmation par aspects, nous permettant ainsi d'écrire SLCA + Aspects = AAs. SLCA peut être considéré comme l'approche choisie pour de la composition, tandis que les aspects ont pour objectifs de traiter la préoccupation de l'adaptation.

Dans la programmation par aspects une application se découpe en **aspects de base** et **aspects non-fonctionnels**.Les premiers définissent les fonctionnalités de l'application tandis que les seconds décrivent des préoccupations transverses comme la sécurité ou encore la persistance. Un aspect se compose de points de coupes et de greffons. Les points de coupes indiquent où injecter du code tandis que les greffons décrivent le code à tisser. La fusion entre ces différents aspects s'opère au niveau de points de jonctions, qui sont les « ancres » sur lesquels s'attachent deux aspects, par un tisseur. La programmation par aspects consiste donc en des injections de code dans du code déjà existant, ce tissage pouvant être réalisé aussi bien lors d'une phase de compilation qu'à l'exécution.

Dans le cadre des AAs ses notions sont toujours valables mais avec quelques nuances, un greffon décrit ici une reconfiguration architecturale d'un assemblage grâce au DSL ISL4WComp. Les points de coupes sont exprimés à l'aide d'expressions régulières décrivant les composants d'un assemblage où appliquer une modification structurelle. Il devient alors possible de reconfigurer dynamiquement à l'exécution un assemblage.

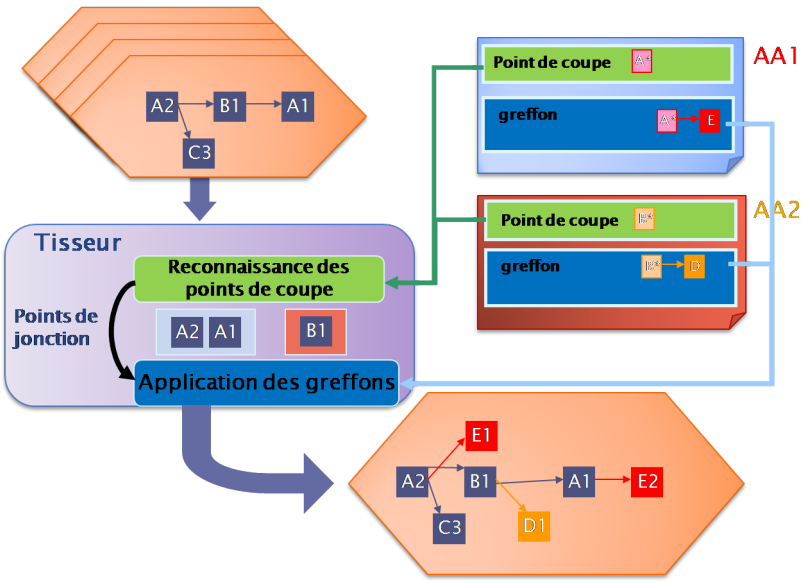

Fig. 4.2 – Application d'aspects d'assemblages

Le processus de tissage des aspects suit le déroulement suivant :

1. Sélection de l'aspect à appliquer
2. Reconnaissance des points de coupe
3. Fusion des aspects
4. Transformation des aspects en requêtes élémentaires de reconfiguration

La construction d'une application peut alors se voir comme l'application successive d'AAs construisant au fur et à mesure qu'un assemblage se modifie l'application finale (l'apparition de nouveau composants déclenchant le tissage de nouveaux aspects). Il est important de noter que la mise en place de plusieurs aspects en parallèle peut être conflictuelle, un mécanisme de fusion des aspects à donc été mis en place. Mécanismes peu fréquents parmi tous les intergiciels étudiés.

Middleware	**Résolution des conflits**
Gaia	X
RCSM	
CARMEN	X
Amigo	
Aura	
CAMidO	
CARISMA	X
Oxygen	
SAFRAN	
CORTEX	
WComp	X

4.3 Sensibilité au contexte

La plupart des mécanismes de prise en compte du contexte présentés précédemment ne sont pas encore présents dans WComp, seule la découverte dynamique autonome de nouveaux services ainsi qu'une interface graphique de configuration pour les AAs existent. La découverte dynamique repose sur UPnP, lors de la découverte de services UPnP des instances de proxies sont générés dans un assemblage. Par la suite les AAs sélectionnés par un utilisateur peuvent s'appliquer si les composants nécessaires sont présents. Il faudra donc par la suite ajouter dans le processus de prise en compte du contexte par WComp une phase d'évaluation des conditions contextuelles afin de savoir quand déclencher un aspect en particulier. Pour être le plus réactif possible un tel mécanisme doit être averti au plus vite des changements opérés dans son environnement. Pour une meilleure efficacité il doit donc être abonné à un mécanisme de publish/subscribe ou à des événements.

Chapitre 5

Mise en oeuvre et choix techniques

Comme nous venons de le voir, WComp ne prend actuellement que faiblement en compte le contexte et ses variations. Il dispose par contre d'un mécanisme lui permettant de réaliser des adaptations dynamiquement avec une approche décentralisée : les Aspects d'Assemblages (AAs). Il faut donc désormais leurs ajouter une dimension contextuelle. Celle-ci s'exprimera grâce à des conditions qui doivent tenir compte de deux niveaux de déclenchement : le contexte et donc l'observation de l'environnement ou un déclenchement explicite par l'utilisateur ; ces deux niveaux pouvant entrer en conflits.

5.1 Schémas contextuels et pertinence de l'infrastructure

Une première approche avait été proposée en 2006 afin d'étendre WComp vers une plateforme sensible au contexte. Il s'agissait de la notion de schémas contextuels qui définissait une association de conditions contextuelles et d'aspects d'assemblage [5]

Schéma contextuel=<conditions contextuelles, Aspects d'assemblages> [5]

Il s'agissait alors d'utiliser un ensemble de conditions contextuelles comme filtre pour sélectionner des aspects d'assemblages et par conséquent de spécifier quand déclencher une adaptation en fonction du contexte. Une entité joue alors le rôle de filtre contenant toutes ces conditions. Ces travaux ont étés validés par une implémentation d'Alin Drimus dans le cadre d'un stage à Polytech'nice.

Cette approche, comme les plateformes applicatives, repose donc sur une architecture centralisant la connaissance du contexte et les conditions contextuelles. Or comme nous l'avons évoqué précédemment les contraintes de l'informatique ambiante et en particulier la nécessité de travailler dans un environnement « multi » (multi-utilisateur, multi-dispositifs ...) rend cette architecture difficilement envisageable. Mais ces problèmes avaient déjà étés aperçus et ces travaux reposent sur la notion de contexte pertinent.

Dans une vision classique de la prise en compte du contexte, lors de la création d'une application le développeur établi pour une situation donnée ce qu'est le contexte pertinent et en défini sa structure. A partir de là il créé un ensemble d'observers qui auront pour tâche de mettre à jour ce contexte pertinent. Malheureusement ceci dans le cadre de l'informatique

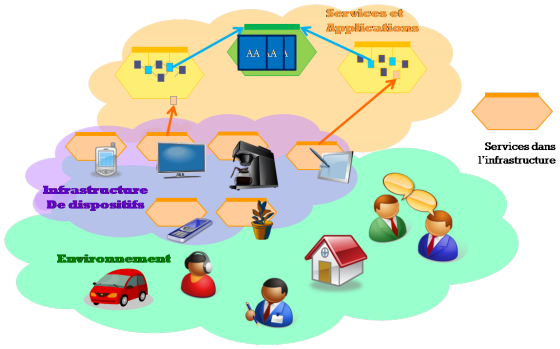

FIG. 5.1 – Architecture centralisée

ambiante n'est plus viable. La haute variabilité de l'environnement, qu'implique en particulier la mobilité, requiert de la part du système une capacité à s'y adapter comme nous l'avons déjà vu. Ainsi un développeur à priori ne peut pas spécifier tous les contextes pertinent, et par conséquent toutes les infrastructures logicielles possibles, pour tous les services possibles ceux-ci n'existant pas forcément à priori. Il est donc primordial pour une bonne sensibilité au contexte de considérer l'évolution de l'infrastructure logicielle d'une application et ce le plus dynamiquement possible.

Les intergiciels que nous avons étudiés précédemment prennent plus ou moins en compte les évolutions de leurs infrastructures logicielles. Si SAFRAN [11] et CAMidO [31] ne le permettent pas, à l'inverse RCSM en à la capacité, il lui est permis de générer de nouveaux composants en fonction des changements survenant dans son contexte opérationnel. Toutefois pour que ces derniers soient utilisables par le middleware un développeur doit créer et leurs associer une interface CA-IDL. D'autre part les plateformes Gaïa [12] et Aura [23] sont également sensibles à leur environnement d'exécution. Les entités composant l'infrastructure logicielle sont stockées dans un annuaire leurs permettant alors d'offrir différents services et ainsi de les choisir pour un comportement souhaité. Amigo [9] dispose également d'un tel mécanisme mais offre plus de possibilités puisque les entrées de l'annuaire sont triées en fonction de leur sémantique et plus particulièrement par leurs similitudes sémantiques. Enfin d'autres intergiciels pour tenir compte des évolutions de leur infrastructure logicielle modifient leurs architectures. Dans le cas d'Oxygen [7] il s'agit de faire évoluer la topologie du réseau N21, pour CORTEX [32] de modifier un assemblage d'objets sensibles. Pour ces derniers, être sensibles aux changements intervenant dans leur contexte opérationnel implique une adaptation structurelle qui peut engendrer une modification du comportement de l'application.

Si avoir la capacité de prendre en compte les évolutions de l'infrastructure logicielle est essentiel, l'informatique ambiante requiert de la part de ces mécanismes une forte dynamicité. En conséquent des mécanismes de découverte et de recherche dynamique de nouveaux services jouent un rôle primordial pour une prise en compte dynamique de l'infrastructure logicielle. Parmi les intergiciels étudiés tous n'ont pas la même approche face à cette problématique. Dans CAMidO [31] et SAFRAN [11] les nouvelles entités sont intégrées par le développeur statiquement. D'autres proposent des approches de découverte subissant quelques restrictions, Oxygen [7] par exemple ne s'intéresse qu'a des dispositifs spécifiques : les H21. CARISMA [15] ajoute la notion de qualité de service à sa découverte mais ne fonctionne en conséquence

qu'avec des ressources locales. Enfin CARMEN [21] ne peut découvrir que quelques ressources disponibles à travers l'internet. Toutefois quelques autres intergiciels font de la découverte dynamique autonome comme CORTEX [22], Amigo [9], Aura [23] ou encore WComp [33].

Nous voyons donc que peu d'intergiciels sont à la fois capables de prendre en compte l'évolution de l'infrastructure logicielle et de faire de la découverte et de la recherche de nouveaux services pour y ajouter de la dynamicité. En effet seules les plateformes CORTEX, Amigo, Aura et WComp regroupe ces deux critères.

Mais le problème présenté précédemment subsiste même pour les middlewares possédant les deux capacités évoquées précédemment. Dans un environnement ubiquitaire il reste plus que concevable (puisque cela est déjà évoqué par Weiser dans [1]) d'envisager des infrastructures très conséquentes avec de nombreux dispositifs en jeux. L'infrastructure à considérer devient alors trop volumineuse pour être entièrement considérée. L'idée proposée ici est de réaliser un filtrage et de conserver les entités considérées comme pertinente. Il convient alors de considérer trois types de contexte. Le **contexte global** qui comme nous le disions n'est pas appréhendable puisqu'il s'agit de considérer le monde entier. Il est donc nécessaire de le filtrer. Le premier médium pour réaliser un filtrage est de considérer le **contexte local**, c'est-à-dire les informations qu'il est permis d'obtenir et les entités accessibles. Cependant cela reste potentiellement trop large. Intervient alors la notion de **contexte pertinent** qui est défini par un réel filtrage de ces informations et entités. L'architecture proposée par Alin Drimus pour répondre à cette problématique, dans le cadre d'un stage au sein de Polytech'nice, appliquée à WComp, mais pouvant également être mise en oeuvre sur d'autres plateformes, repose sur quatre étapes regroupées dans un designer :

1. Modélisation du contexte à la main sous forme d'un arbre (XML)
2. Associer des paramètres contextuels aux dispositifs
3. Abonnements à des observers, les données observées ont été modélisées précédemment
4. Création, évaluation de règles de filtrage pour l'apparition et la disparition de dispositifs dans le contexte d'un assemblage

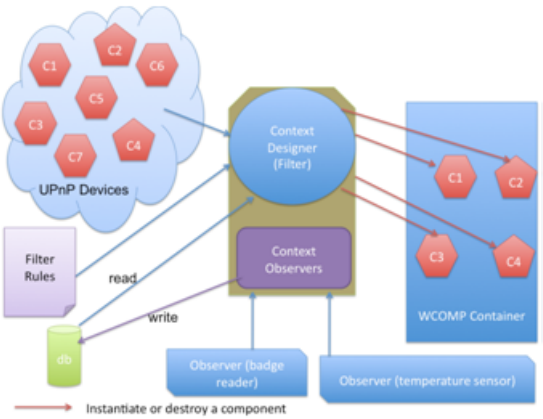

FIG. 5.2 – Architecture du module de filtrage

Ces travaux permettent à WComp de prendre en compte le contexte et d'être comparés aux plateformes applicatives du domaine. Mais la notion de schéma contextuel repose toujours sur une architecture centralisée même si seule l'infrastructure pertinente est considérée.

5.2 Vers une évolution des AAs

Comme nous l'avons évoqué nous souhaitons désormais avoir une approche architecturale décentralisée. D'autre part nous avons vu que les aspects d'assemblages sont déjà des entités réutilisables pour l'adaptation et qu'il est déjà envisageable d'avoir un objet intelligent fournissant des AAs à d'autres objets à la manière des systèmes pair à pair. Finalement il s'agit d'étendre ces aspects afin qu'ils décrivent désormais à la fois « comment » adapter mais aussi selon quelles conditions : « quand » adapter. Plus précisément il s'agit d'étendre les points de coupe des AA. En effet les points de coupe plus que le « où » définissent en programmation par aspects les points d'intérêt d'une application pour le tissage d'un greffon, à savoir jusqu'à présent dans notre cas des composants. Or dans le cadre d'une adaptation au contexte les points d'intérêts ne sont plus seulement la présence de composants mais plus largement l'observation du contexte.

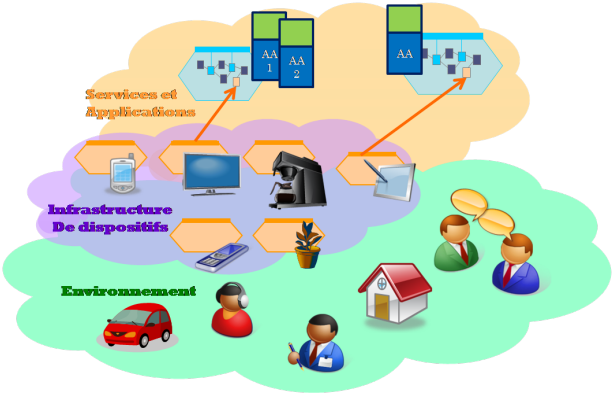

FIG. 5.3 – Une architecture décentralisée

L'étude des différents intergiciels nous à permis d'établir les différentes phases nécessaire à une bonne prise compte du contexte. Que nous pouvons fortement simplifier comme suit :

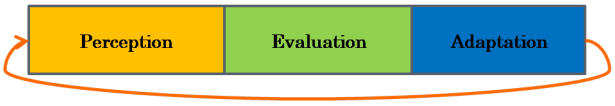

FIG. 5.4 – Cycle simplifié de la prise en compte du contexte

Comme nous l'avons vu les AA permettent de réaliser l'étape d'adaptation, il conviendra donc d'incorporer dans ces extensions des points de coupe les descriptions des deux autres : perception et évaluation. Ces phases nous permettront de décider quand appliquer une adaptation.

Un adaptation est mise en oeuvre dans une situation donnée qui est elle-même déterminée grâce à la phase d'évaluation des informations contextuelles. On passe d'une situation à une

autre lorsque des changements interviennent dans le contexte, changements obtenus quand à eux grâce à la phase de perception du contexte. Nous pouvons donc représenter ce mécanisme de décision sous la forme d'un graphe d'état dont les entrées sont des changements de contexte et les sorties des adaptations. On passe d'un état (une situation) à un autre en fonction de changements de contexte.

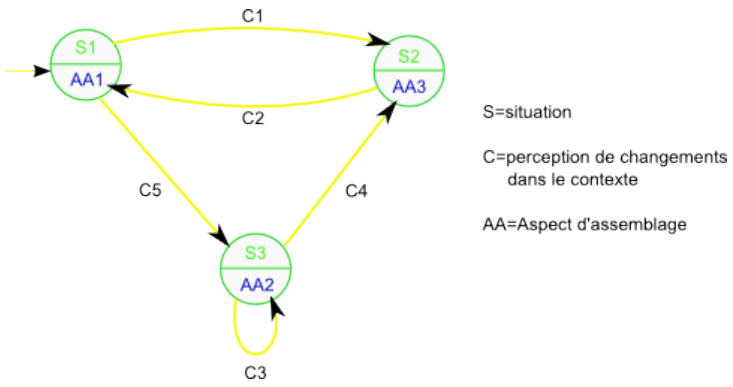

FIG. 5.5 – Machine de Moore et prise en compte du contexte

Nous devons donc être capables d'exprimer dans les extensions des points de coupes des AA à la fois ces changements mais aussi être capable d'identifier une situation contextuelle. Dans WComp les observations des changements dans le contexte sont obtenues par l'intermediaire de dispositifs, se trouvant dans l'environnement du système, chargés de collecter des informations contextuelles : les observers. Ces objets avec qui il est possible de communiquer grâce aux web services pour dispositifs propagent des événements qui indiquent les variations de contexte. Les points de coupe doivent donc considérer ces événements. Ainsi actuellement un point de coupe est une fonction qui prend en entrée une description de composants et fourni en sortie des points de jonctions. Ils permettent ainsi de designer « où » appliquer un greffon et par conséquent d'en dupliquer un en plusieurs endroits.

F(C1,C2,C3) -> C'1, C'2, C'3

Ils doivent évoluer vers :

F(C1,e,C2) -> C'1,C'2,C'3

Cependant la simple reconnaissance de compositions d'événements n'est pas suffisante pour identifier une situation. Plus que l'information de l'apparition d'un changement dans l'environnement il faut pouvoir évaluer la situation dans laquelle se situe le système. L'occurrence d'un événement indiquant que la température augmente n'est pas forcément suffisante à l'application d'une reconfiguration il faut peut être vérifier qu'il fait nuit et que la pièce est vide pour cela. Il faut donc être capable d'identifier une situation, ce que nous proposons de réaliser sous la forme d'un ensemble de conditions. Plus simplement il aurait été possible d'intégrer ces conditions directement dans les greffons puisque cela est déjà possible grâce à la présence d'opérateurs de conditionnelles et par conséquent en générant certains composants dans l'assemblage (par exemple des composants if ou seuils). Toutefois cette approche souffre de deux inconvénients : elle génère potentiellement une quantité de composants importantes et l'évaluation des conditions se fait donc par l'application et non plus pour la sélection de l'aspect. Ces conditions n'adressent toutefois pas exactement le même problème, dans les greffons ces

conditions définissent le comportement de l'application tandis que dans les points de coupe ils régissent le comportement de l'adaptation. Ce dernier choix colle donc mieux à nos attentes.

Le Framework E-AOP [35] propose également de faire de la programmation par aspects en définissant les points de coupes comme des séquences d'événements. Ainsi à une séquence est associée une action à réaliser. Les aspects sont déclenchés avec l'évolution de la trace de l'exécution du programme. Pour cela les objets sont wrappés afin d'émettre des événements lors d'appels et de retours de méthodes par un préprocesseur. Un moniteur observe les événements émis et les propagent à tous les aspects. Quand il est appelé le programme de base est arrêté tant que tous les aspects n'ont pas étés testés, il traite les aspects séquentiellement. Enfin leur définition des aspects offre la possibilité d'encapsuler les aspects entre-eux. L'approche que nous proposons se différencie de celle proposée ici car elle met en jeux services et événements, et plus particulièrement des événements de web services pour dispositifs. Dans notre cas nous proposons donc de ne pas seulement observer la trace d'exécution du programme mais de prendre en compte également les événements relatifs à l'observation de l'environnement. D'autre part les aspects d'assemblages proposent quelques différences par rapports aux aspects classiques en conservant par exemple la propriété d'encapsulation.

Nous devons donc exprimer dans les extensions des points de coupe non seulement les événements obtenus grâce aux observer mais aussi un ensemble de conditions dont l'évaluation nous permettra d'identifier la situation du système.

5.2.1 Logiques et mécanismes de décision

Afin d'exprimer ces événements issus de l'observation du contexte ainsi que les conditions nous permettant d'identifier la situation dans laquelle se trouve un système, différents types de logiques existent. Plus que la seule prise en compte de l'infrastructure pertinente il s'agit ici d'ajouter un coté opportuniste à la prise de décision. C'est-à-dire considérer les changements apparaissant dans le contexte et y être au maximum réactif tout en conservant toujours notre approche décentralisée. Ainsi plusieurs types de logique plus ou moins puissants et complexes à mettre en oeuvre sont utilisables. On note entre autre :

Logique des propositions : Une proposition est du langage ordinaire considéré du point de vue formel : un énoncé est soit vrai soit faux mais pas les deux. Il s'agit d'une des logiques les plus simples. Une proposition contient juste des variables et des connecteurs logiques, pas de quantification, pas de fonctions, pas de prédicats. Par exemple : AvB->C

Logique premier ordre : Elle ajoute à la logique des propositions des quantificateurs, des relations (qui peuvent être d'arité variable), des fonctions ainsi que des constantes. La logique du premier ordre est la logique des formules usuelles, avec la contrainte que les variables représentent toutes des objets du même type.

Inférence bayésienne : On nomme inférence bayésienne la démarche logique permettant de calculer ou réviser la probabilité d'une hypothèse. Cette démarche est régie par l'utilisation de règles strictes de combinaison des probabilités, desquelles dérive le théorème de Bayes.

Raisonnement non-monotone : Le raisonnement non monotone est un processus d'in-

férence basé sur des informations partielles ou de véracité incertaine et dont les conclusions peuvent être rétractées avec l'ajout de nouvelles informations : les conclusions ne croissent pas nécessairement de façon monotone avec l'ajout de nouvelles connaissances.

Raisonnement floue : Un ensemble flou est un ensemble dont la fonction d'appartenance prend des valeurs dans tout [0,1] et non plus simplement 0 ou 1. Les fonctions de transfert permettent de définir l'appartenance à un groupe, elles sont définies en fonction du problème. Un système de règles floues permet de décrire sous forme de règles linguistiques une fonction de transfert. Et par conséquent l'appartenance à un groupe selon un certain degré de validité. En logique floue on peut décrire une eau encore un peu froide et qui commence à être tiède.

On dénombre trois grandes familles de mécanismes utilisés par les différents middlewares étudiés, chacune d'entre elles s'associant à un type de logique. Ainsi CARMEN [21], Amigo [26], CAMidO [16], Oxygen [7] ou encore SAFRAN [11] utilise le modèle ECA (event-condition-action) tiré des bases de données dans leur phase d'évaluation. Lors de l'occurrence d'un événement on évalue des conditions qui, si elles sont validées, déclenchent une action (bounas). L'expression des conditions dans ces intergiciels met en jeu de la logique des propositions. RCSM [24] et Cortex se proposent également d'utiliser des règles ECA mais dont les conditions peuvent se composer d'opérateur plus évolués, dans le cas de Cortex le langage CLIPS permet d'écrire de vrais programmes [34]. Ils se rapprochent donc plus de la logique de premier ordre. Enfin les intergiciels Gaia [12] et CARISMA [15] utilisent en plus de cela de la logique floue, qui de par la souplesse qu'elle autorise permet de traiter les problèmes d'incertitude des données contextuelles.

Middleware	**Mécanismes de décisions**
Gaia	ECA+logique floue
RCSM	ECA+ [1]
CARMEN	ECA
Amigo	ECA
CORTEX	ECA+
Aura	ECA
CAMidO	ECA
CARISMA	ECA+logique floue
Oxygen	ECA
SAFRAN	ECA

Dans le cadre de WComp si toutes ces pistes restent toutefois envisageables nous nous orientons vers un système à base de règles ECA qui collent bien aux besoins évoqués précédemment, les événements (E) décrivent les changements de contexte qui nous amène à un état (une situation) identifié par des conditions (C) pour déclencher une adaptation (A) . Couplé à la notion d'infrastructure pertinente on obtient alors des règles :

IN infrastructure WHEN event IF condition THEN action

Règles qui utiliseront entre autre de la logique du premier ordre mais qui pourront par la suite évoluer vers d'autres pistes.

[1] ECA+=ECA évolué

5.2.2 Changements de situation

A un même moment de multiples changements peuvent intervenir dans le contexte d'une application, il convient donc d'intégrer à ces extensions des points de coupe la possibilité d'exprimer des événements complexes, des compositions d'événements. Différents opérateurs peuvent nous permettre d'exprimer ces événements complexes, nous devons pouvoir les composer tout en gérant les problèmes de synchronisation. D'autre part afin d'accéder aux événements fournis par le container via l'interface structurelle et permettant d'accéder aux informations relatives à l'assemblage il est nécessaire d'ajouter un opérateur jouant ce rôle. L'utilisation de patrons d'événements peut mener à des conflits d'adaptation, en effet plusieurs aspects peuvent s'appliquer pour une séquence d'événements et par conséquent aboutir à un résultat non attendu. Par exemple soit la série d'événements e1 ;e2 ;e3 et les aspects se déclenchant à partir des patrons suivant : A1(e1,e2) et A2(e1,e2,e3). Il reste envisageable que l'aspect A1 réalise une adaptation non attendue alors que A2 soit l'aspect à appliquer. Cependant les deux patrons sont validés. Afin d'éviter ce problème nous ajoutons l'opérateur not exprimant la non occurrence d'un événement.

Opérateurs	**Description**
;	Exprime la séquence
|	Exprime un ordre indifférent (parallèle)
Structural	Permet d'accéder aux événements propagés via l'interface structurelle
not	Exprime la non occurrence d'un événement

Toutefois ces opérateurs pourront faire l'objet d'une recherche plus approfondie par exemple en se basant sur les propriétés CARE [36] pour les interfaces multimodales. Dans ces propriétés on trouve la Complémentarité de deux modalités, notion que l'on peut retrouver avec nos opérateurs « ; » et « | ». L'affectation, il faut passer par un état A pour atteindre le B, que l'on retrouve avec l'opérateur « ; ». La redondance pour désigner le fait qu'une modalité peut être plus expressive qu'une autre, ce point n'est pas abordé ici. Enfin l'équivalence supprime la notion d'ordre que l'on avait avec l'affectation, comme le propose l'opérateur « | ». D'autre part une autre piste de poursuite reste le couplage de ces propriétés avec les relations de Allen [37] qui décrivent de la logique des instants. Des travaux sur la multimodalité vont également dans ce sens [38].

5.2.3 Identification d'une situation

Nous proposons donc pour identifier une situation d'exprimer un ensemble de conditions dans les points de coupe permettant ainsi de la caractériser. Afin de pouvoir exprimer les différentes conditions dans une close *If* nous devons posséder les opérateurs classiques de logique booléenne et de comparaisons. Nous proposons donc la grammaire suivante nous permettant d'ajouter de la sémantique aux points de coupes :

Vocabulaire :
<variable> ::= (nom du filtre) paramètres ;[|autre filtre]
<operateur> ::= '==' | '<' | '>' | '>=' | '<='
<operateurBooleen> ::= '&&' | '||' | 'not'
<operateurEvenement> ::= '|' | ';' | 'structural' | 'not'
<exp> ::= <composant>.<méthode> [|<operateur> <composant>.<méthode>]

```
|<exp><operateurBooleen><exp>
<event> ::= <composant>.<événement>
|<event><operateurEvenement><event>
Grammaire:
When <event> {
if (<exp>){
<variable>
...
}
}
```

5.2.4 Exemple d'Aspects d'assemblages étendus

En appliquant cette grammaire a l'exemple tiré de [5] nous vérifions bien que nous avons la possibilité d'exprimer l'équivalent des schémas contextuels dans les aspects d'assemblages étendus, et par conséquents les conditions contextuelles à l'origine du déclenchement d'une adaptation. Considérons le schéma suivant :

```
Conditions contextuelles : 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() }
}
```

Nous exprimons donc les conditions contextuelles de cet aspect à travers les points de coupe suivants :

```
When structural.^New_Bean{ /*si un nouveau composant
time := /horloge/ est à proximité contextuelle*/
If((time.getHour() >= 12)&&(time.getHour() <= 14)){
Joystick := /joystick[[:digit:]]/ /* les composants requis
ecrantv := /ecrantv[[:digit:]]/ sont biens dans la
}} zone contextuelle*/
```

Nous voyons donc l'intérêt du mot clé structural qui permet d'accéder aux événements originaires de l'interface structurelle des containers. Nous voyons également que les patrons d'événements de la clause when ne suffisent pas toujours à décrire une situation et doivent pouvoir

être complétés avec d'autres conditions. Toutefois cette clause n'est pas toujours obligatoire, la composition d'événements peut être suffisante comme dans cet exemple :

> Considérons une pièce équipée d'un système de localisation ,d'une lampe, de capteurs sur la porte et d'un écran.
> ***Conditions contextuelles*** : Lorsque l'utilisateur entre dans la pièce et ferme la porte la lampe s'allume et son emploi du temps sur l'écran.

Nous exprimons donc les conditions contextuelles de cet aspect à travers les points de coupe suivants :

```
Location := /ubisense/
When location.^Inside | door.^Closed {
light:=/light[[:digit:]]/
ecran:=/ecrantv[[:digit:]]/
}
```

5.3 Conséquences

Les aspects d'assemblages étendus sont donc des entités réutilisables qui nous permettent d'avoir une approche décentralisée de la prise en compte du contexte. Ainsi un objet peut apporter ces propres aspects et les transmettre à d'autres objets, d'autres services composites. Lorsqu'un aspect étendu est reçu par un service composite ce dernier aura pour première tâche de vérifier la présence dans son infrastructure de dispositifs des observers nécessaire à son évaluation. C'est-à-dire qu'il a bien la possibilité d'accéder aux informations décrites dans les points de coupe (événements et conditions). Les aspects étendus respectent donc le cycle de vie suivant :

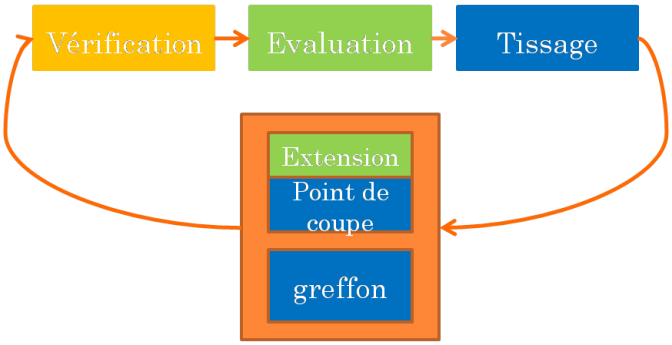

FIG. 5.6 – Cycle de vie des AA étendus

Toutes ces modifications auront donc des conséquences aussi bien sur les mécanismes d'observation du contexte que sur les mécanismes de tissage.

5.3.1 Sur la prise en compte du contexte

Notre démarche pour ajouter une nouvelle dimension contextuelle dans les aspects d'assemblages consiste donc entre autre en l'observation d'événements permettant de décrire une

situation dans les points de coupes. Afin de pouvoir se baser sur ces derniers il est nécessaire pour un designer d'AA d'interagir avec l'interface fonctionnelle de containers. Par conséquent la présence de composants sondes pour chacun de ces événements est requise. Ainsi à toute description <composant.événement> d'un point de coupe doit être associée une sonde pour l'observation de cet événement. Le couple <composant.événement> est nécessaire car des formulations trop génériques ne permettent pas de connaître l'origine d'un événement et ne garantissent pas leur unicité.

Afin de générer ces sondes deux solutions restent envisageables : utiliser des scripts pour modifier l'assemblage ou utiliser des « greffons d'observation du contexte ». Les aspects dont l'objectif est justement de réaliser des reconfigurations d'un assemblage offrent des propriétés comme l'associativité, la commutativité et l'idempotence que l'utilisation de simple scripts ne nous garanti pas. Il serait donc intéressant d'être capable de générer dynamiquement de tels greffons d'observation du contexte. D'autre part, une fois cet advice généré, il pourrait être associé à l'AA d'origine comme métadonnées. Ces greffons d'observations prendront la forme qui suit dans le cas d'un point de coupe offrant la clause when de la forme :

```
POINTCUT:
observer:=/light/
ADVICE:
schema AA_PROBE(observer):
sonde : mWcomp.BasicBeans.EventProbe'
observer.^Status_Evented_NewValue->(
sonde.NotifyContainer()
)
```

Un cycle de mise en place de ces greffons pourrait donc être le suivant :

1. Sélection des aspects à considérer
2. Génération des greffons d'observation du contexte
3. Tissage de ces greffons
4. Observation du contexte

Enfin comme nous l'avons évoqué précédemment les observers peuvent réagir à de trop faibles variations et déclencher malencontreusement des adaptations non prévues. Ainsi par exemple la luminosité d'une pièce peut baisser très faiblement, cette modification peut être observée par un capteur de luminosité qui propagera alors l'événement correspondant au designer d'AA. Ce dernier serait alors susceptible de déclencher une adaptation, toutefois l'ajout de closes if dans les points de coupes permettent de parer à ce problème et donc de filtrer ces trop faibles variations.

5.3.2 Sur les mécanismes de tissage

Ces extensions des points de coupes nécessitent d'apporter des modifications aux différents mécanismes mis en jeu dans les aspects d'assemblages, et plus particulièrement au tissage. Jusqu'à présent les points de coupes sont exprimés en AWK, les extensions à apporter aux AA requièrent d'y ajouter de la sémantique. La nouvelle grammaire définie précédemment nécessite un nouvel interprète. Ainsi le tisseur doit être capable de prendre en compte les événements

fournis par les différentes entités de l'infrastructure logicielle mais aussi les informations relatives à l'assemblage de composant. Il faut donc que le designer d'aspects d'assemblage ait la capacité de prendre en compte les interfaces structurelles et fonctionnelles. D'autre part afin de pouvoir évaluer les clauses if des points de coupes, le designer doit être capable de faire appel aux méthodes des composants d'un assemblage. Pour accéder aux événements de l'interface fonctionnelle il faut associer aux entités qui les propagent (nos observers) des composants sondes.

Comme souvent lorsque l'on traite avec les événements il est important de s'intéresser à leurs problèmes de synchronisation. Dans notre cas les compositions d'événements pour se baser sur des séquences d'événements requièrent un tel mécanisme. Le nombre d'événements émis pouvant être élevé, le designer doit être capable de tous les prendre en compte ainsi que leur ordre d'émission. Pour cela une structure de file d'attente semble appropriée. Toutefois l'ordre de réception des événements n'est pas forcement suffisant pour assurer leur bon ordonnancement. Pour ce faire il peut être envisagé d'utiliser les numéros de séquence des événements UPnP afin de les trier. En effet avec UPnP chaque événement est envoyé avec un numéro de séquence qui est incrémenté à chaque événement afin que les points de contrôle soient certains de les traiter dans l'ordre [39]. D'autre part la file d'attente permet de conserver les événements lorsque le tisseur se bloque pour appliquer un aspect. Toutefois ce point reste néfaste à une bonne réactivité.

Actuellement lorsque les composants permettant de déclencher un aspect sont présents, celui-ci ne s'applique qu'une fois lorsque tous les critères sont remplis. De la même manière un aspect déjà mis en oeuvre ne doit pas se réappliquer lorsque les conditions, et donc les bons événements, sont remplies. Il convient donc de conserver le même mécanisme qu'auparavant. On voit également que face à la multiplicité des événements qu'il est acceptable d'envisager, il conviendra de les filtrer, les capteurs pouvant réagir à de trop faibles variations par exemple. D'autre part les patrons d'événements pouvant être des séquences il faut donc conserver les événements qui se trouvent dans la FIFO tant qu'ils vérifient une partie d'une séquence d'un aspect. Parallèlement à cela il faut que ces événements faisant potentiellement partie d'une séquence soient évalués lors de leur arrivée par les autres aspects. Et par conséquent, il faudra gérer les problèmes de composition des points de coupe que la décentralisation implique. Enfin il conviendra de gérer les conflits de sélection d'aspects lors d'un déclenchement d'aspects simultané par des utilisateurs et par observation du contexte.

Chapitre 6

Conclusion

Nous avons introduit dans ce rapport une approche originale pour la prise en compte du contexte, et plus particulièrement des mécanismes de déclenchement d'adaptations, en se basant sur une architecture décentralisée. Cette approche appliquée à la plateforme WComp sera donc à l'origine de modification dans les aspects d'assemblage, et plus précisément dans les points de coupe qui doivent être enrichis pour s'apparenter à un mécanisme décisionnel ECA (Event-Condition-Action) originaire des bases de données actives. Les aspects d'assemblages sont désormais des entités réutilisables qui peuvent être échangées entre services composites. Ils décrivent à la fois l'adaptation à réaliser mais aussi les conditions sous lesquelles appliquer cette adaptation. Ces dernières prendront alors la forme de compositions d'événements pour une meilleure réactivité et de clauses de conditionnelle pour une meilleure expressivité. Toutes ces modifications auront des conséquences sur les mécanismes de tissage et d'observation du contexte ainsi que sur le cycle de vie des aspects. La composition/fusion des points de coupe amenés par différents dispositifs en est un exemple et pourra faire l'objet de recherches. D'autre part il sera necessaire dans le futur d'envisager les conflits que peuvent engendrer des décisions utilisateur par rapport à celles du mécanisme de l'intergiciel. Enfin nous avons également vu qu'une étude des propriétés CARE et des relations de Allen permettrait d'étendre l'expressivité des compositions d'événements.

Suite à mon projet de fin d'étude qui avait déjà pris une orientation recherche, ce stage au sein de l'équipe Rainbow m'a permi de continuer à me forger une culture recherche dans l'optique de ma poursuite d'étude. Mais aussi de poser les bases de mes travaux futurs grâce à mes travaux bibliographique et aux premiers résultats trouvés.

Annexe A

Intergiciels existants

Amigo[8] pour AMbient Intelligence to GO est un middleware open-source orienté service. Il a pour but de proposer des mécanismes pour intégrer dynamiquement des systèmes hétérogènes et de les faire communiquer entre eux. De nombreux protocoles existent et sont intégrés dans l'intergiciel pour la découverte automatique de nouveaux services. Il propose également des mécanismes pour prendre en compte le contexte ainsi qu'un Framework pour le développement de services « amigo-aware ».

CORTEX[10] pour CO-operating Real-time senTient objects à pour objectif d'aider à la conception d'applications sensibles au contexte et adresse plus particulièrement le problème de la réactivité. Il fournit à travers un nouveau modèle d'objets sensibles des mécanismes de gestion des informations contextuelles. Il est implémenté sur openCOM qui est lui-même un intergiciel basé sur la technologie COM de Microsoft. Ce qui lui permet d'utiliser la réflexivité et des objets pour effectuer ses reconfigurations. Une application dans CORTEX se compose d'un réseau d'objets sensibles et de déclencheurs reconfigurable.

CAMidO[31] est un intergiciel qui a pour but de faciliter le développement d'applications sensibles au contexte. Il s'adresse plus particulièrement au problème de la séparation des préoccupations d'adaptations et des mécanismes métier ainsi qu'a réaliser à la fois des adaptations réactives et proactives. Pour réaliser ses adaptations CAMidO utilise un compilateur génèrant du code qui sera inséré dans un contrôleur qui gère les adaptations au niveau composant.

Aura[6] est un intergiciel orienté tâches qui se base sur la notion de « personal aura » et s'intéresse plus particulièrement au problème de la migration de la tâche utilisateur. Une aura personnelle joue le rôle de proxy pour l'utilisateur qu'elle représente et se charge d'essayer de l'aider à réaliser sa tâche dans les meilleures conditions. Contrairement à la plupart des intergiciels aura ne cherche pas à aider au développement d'applications sensibles au contexte mais cherche à utiliser celles existantes. Les changements dans le contexte sont notifiés par des événements et peuvent engendrer des changements dans la tâche utilisateur.

Gaia[20] offre la possibilité de développer des applications sensibles au contexte centrées utilisateurs, applications qui peuvent également évoluer dans un environnement multi-dispositifs et être « resources-aware ». Il a ainsi pour objectif d'aider à la conception d'espaces ubiquitaires. Pour ce faire Gaia se comporte comme un système d'exploitation en proposant

un système de fichiers mais aussi une couche de communications, la gestion des ressources et leurs allocations, les opérations d'entrées-sorties ou encore la gestion de l'exécution des programmes. Ce système reste cependant relativement centralisé.

Oxygen[7] est orienté interactions hommes machines et propose une solution pour le déplacement de la tâche utilisateur. Il est possible de communiquer avec Oxygen grâce à de la reconnaissance vocale et des technologies de la vision. Le système Oxygen se décompose en trois parties que sont H21, un service d'appareil mobiles, N21 le réseau et E21 l'environnement se composant d'un ensemble de capteurs. Le H21 est un dispositif qui a pour but de regrouper les fonctionnalités de téléphone portable, radio, agenda, connexion internet sans fil, lecteur vidéo et musical. Il est possible de le configurer à distance pour qu'il se mette dans un de ces modes en particulier. La topologie du N21 peut être facilement modifiée.

CARISMA[15] (Context Aware Reflexive Midleware for Mobile Applications) a pour but de fournir des mécanismes permettant la prise en compte du contexte par la définition de politiques de réaction. Dans l'optique de permettre au middleware de modifier son comportement en fonction du contexte à l'exécution, CARISMA utilise la réflexivité associé à des métadonnées. Celles-ci permettent d'obtenir une séparation entre ce que l'intergiciel fait et de quelle manière. Les politiques d'adaptation pouvant être conflictuelles, il dispose d'un système de résolution des conflits.

CARMEN[21] s'adresse a gérer les ressources de réseaux sans fils avec les problèmes de connections intrinsèques à cette technologie. Pour ce faire CARMEN utilise des proxies sous la forme d'agents mobiles qui se trouvent dans l'environnement de l'utilisateur. Ainsi lors d'un de ses déplacements le proxy de l'utilisateur se déplace avec lui dans son nouveau milieu. Ce proxy lui permettra alors d'accéder aux nouvelles ressources de l'environnement, mais lui assure également l'accès aux ressources qu'il souhaite garder de son ancien environnement soit en les copiant, soit en cherchant des ressources semblables dans le nouveau.

SOCAM[40] (Service-oriented contexte aware middleware) propose une architecture pour créer et prototyper des services sensibles au contexte. La conception de cet intergiciel repose sur l'idée de modéliser le contexte en utilisant des ontologies. Il utilise un modèle du contexte commun à tous les dispositifs sous la forme d'une ontologie. Celle-ci se découpe en 2 catégories, une pour représenter le contexte général, qui est fixé une fois puis est utilisé dans différents domaines. La seconde spécifique au domaine (à l'environnement), de bas niveau, permet d'obtenir les caractéristiques d'un environnement comme une pièce... Ce middleware ne réalise pas d'adaptation.

RCSM[41] pour reconfigurable context-sensitive middleware facilite la réalisation d'applications sensibles au contexte dans un environnement ubiquitaire. Il fourni donc le langage Context Aware IDL(basé sur le corba IDL) aux développeurs qui doivent produire une interface sensible au contexte décrivant les éléments importants pour la sensibilité de l'application mais aussi des règles pour son adaptation. En effet RCSM ne se base pas seulement sur des objets mais sur des objets sensibles au contexte. Ceux-ci se compose de la dite interface en plus de l'implémentation de l'objet qui en est indépendante.

SAFRAN[42] se base au dessus de Fractal, il utilise une approche par aspects pour mo-

dulariser le code d'adaptation d'applications mais aussi afin d'utiliser la dynamicité offerte par le tissage. Cet intergiciel tend donc à faciliter le développement d'applications sensibles au contexte mais ne réalise pas d'interprétation de ce contexte afin d'en déduire des informations de plus haut niveau. D'autre part SAFRAN ne permet pas l'adaptation d'applications distribuées et se concentre plus particulièrement sur l'adaptation de la structure des applications.

Annexe B

Définitions & acronymes

- **AA** : Aspect d'assemblage
- **AOP** : Aspect Oriented Programing
- **CAMidO** : Context Aware middleware based on ontology meta-model
- **CARISMA** : Context Aware Reflective mIddleware System for Mobile Applications
- **CARMEN** : Context Aware Resource Management ENvironment
- **CORTEX** : CO-operating Real-time senTient objects :architecture and EXperimental evaluation
- **DSL** : Domain specific language
- **DPWS** : Devices Profile for Web Services
- **ECA** : Event Condition Action
- **Hiérarchie structurelle** : Il s'agit du raffinement d'une structure en ses parties, en s'appuyant sur une relation de composition de celle-ci en ses sous-structures.
- **Hiérarchie fonctionnelle** : réalise une interprétation téléologique (sur l'étude de la finalité) du comportement du système en exhibant les rôles que jouent chacun des composants structurels dans la réalisation de la fonction du système (but pour lequel le système a été conçu). [Approche Multi-modèle et Hiérarchique pour le Diagnostic Hors-ligne des Systèmes Embarqués]
- **ISL** : interaction spécification language
- **IP** : Internet Protocol
- **Intergiciel** : est un logiciel servant d'intermédiaire de communication entre plusieurs applications qui sont généralement complexes ou bien distribuées sur un réseau
- **LDAP** : Lightweight Directory Access Protocol
- **LUA** : Langage de scripts compact pour être embarqué dans des applications.
- **MOP** : Meta Object Protocol
- **OS** : Operating System
- **Ponder** : Policy specification language
- **QoS** : Quality of Service
- **RDF** : Resource Description Framework
- **SCaLaDE** : Service with Context awareness and Location awareness for Data Environments
- **SLCA** : Service Lightweight component architecture
- **SLP** : Service Location Protocol
- **SSDP** : Simple Service Discovery Protocol

- **UDDI** : Universal Description, Discovery, and Integration
- **UML** : Unified Modeling Language
- **UPnP** : Universal Plug and Play
- **URL** : Uniform Ressource Locator
- **WSDL** : Web Service Definition Language
- **XML** : eXtensible Markup Language

Annexe C

Planning

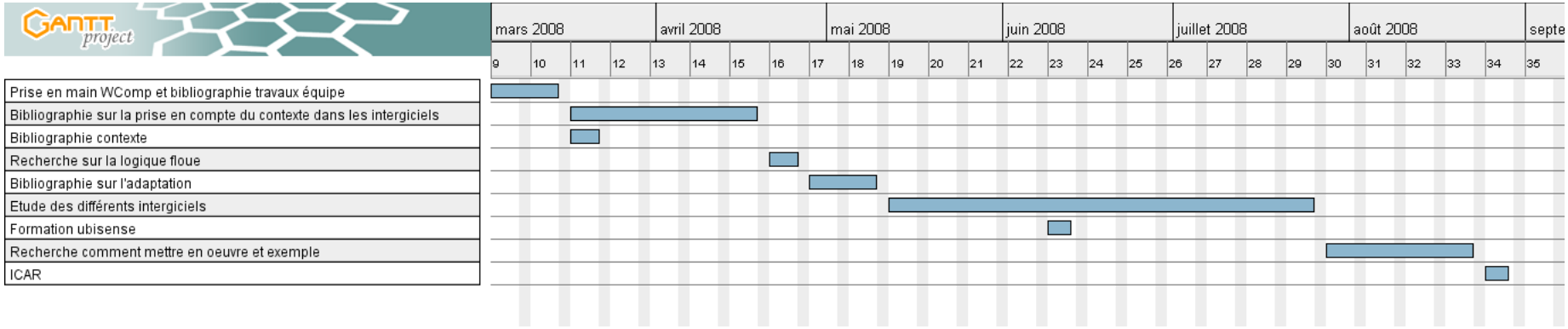

Table des figures

Bibliographie

[1] M. Weiser, "The Computer for the Twenty-First Century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] M. Weiser, "Ubiquitous Computing," *IEEE Computer*, vol. 26, no. 10, pp. 71–72, 1993.

[3] A. Dey, D. Salber, M. Futakawa, and G. Abowd, "An architecture to support context-aware applications," *submitted to UIST*, vol. 99.

[4] M. J. J. Z. S. v. d. m. M. O. F. C. F. W. D. John Strassner, Yan Liu, "Modelling context for autonomic networking," *MUCS 2008*, p. 10, april 2008.

[5] J. TIGLI, D. CHEUNG-FOO-WO, S. LAVIROTTE, and M. RIVEILL, "Adaptation au contexte par tissage d'aspects d'assemblage de composants déclenchés par des conditions contextuelles," *Ingénierie des systèmes d'information(2001)*, vol. 11, no. 5, pp. 89–114, 2006.

[6] J. Sousa and D. Garlan, "Aura : An Architectural Framework for User Mobifity in Ubiquitous Computing Environments," *Software Architecture : System Design, Development and Maintenance*, 2002.

[7] "http ://oxygen.lcs.mit.edu/overview.html,"

[8] D. Sacchetti, Y. Bromberg, N. Georgantas, V. Issarny, J. Parra, and R. Poortinga, "The Amigo Interoperable Middleware for the Networked Home Environment," *6th International Middleware Conference, Workshops Proceedings, Grenoble, France*, 2005.

[9] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers, "Easy : Efficient semantic service discovery in pervasive computing environments with qos and context support," *J. Syst. Softw.*, vol. 81, no. 5, pp. 785–808, 2008.

[10] C. M. P. V. A. C. H. D.-L. M. W. V. C. B. H. R. M. Jörg Kaiser, Cristiano Brudna, "Co-operating real-time sentient objects : architecture and experimental evaluation," tech. rep., Project IST-2000-26031, 2003.

[11] P. David, "Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation," *These de doctorat, Universite de Nantes, Nantes, France, Juillet*, 2005.

[12] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt, "Gaia : a middleware platform for active spaces," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 65–67, 2002.

[13] A. Charfi and M. Mezini, "Aspect-Oriented Web Service Composition with AO4BPEL," *Web Services : European Conference, ECOWS 2004, Erfurt, Germany, September 27-30, 2004 : Proceedings*, 2004.

[14] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "A Taxonomy of Compositional Adaptation," *Rapport Technique numéroMSU-CSE-04-17, juillet*, 2004.

[15] L. Capra, *Reflective Mobile Middleware for Context-Aware Applications.* PhD thesis, University of London, 2003.

[16] N. BEHLOULI, *Ajout de mécanismes de réactivité au contexte dans les intergiciels pour composants dans le cadre d'utilisateurs nomades.* PhD thesis, Université de Nice-Sophia Antipolis, 2006.

[17] J. Pascoe, "Adding generic contextual capabilities to wearable computers," *Proceedings of the 2nd International Symposium on Wearable Computers*, pp. 92–99, 1998.

[18] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," 2000.

[19] J. Coutaz, J. Crowley, S. Dobson, and D. Garlan, "Context is key," *Communications of the ACM*, vol. 48, no. 3, pp. 49–53, 2005.

[20] M. Roman and R. Campbell, "A User-Centric, Resource-Aware, Context-Sensitive, Multi-Device Application Framework for Ubiquitous Computing Environments," *Urbana*, vol. 51, p. 61801.

[21] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli, "Context-aware middleware for resource management in the wireless Internet," *Software Engineering, IEEE Transactions on*, vol. 29, no. 12, pp. 1086–1099, 2003.

[22] H. Duran-Limon, G. Blair, A. Friday, P. Grace, G. Samartzidis, T. Sivaharan, and M. WU, "Context-aware middleware for pervasive and ad hoc environments," *Computing Department, Lancaster University, Bailrigg, Lancaster, UK*, 2000.

[23] J. Sousa and D. Garlan, "From Computers Everywhere to Tasks Anywhere : The Aura Approach," *School of Computer Science, Carnegie Mellon University, http ://www. cs. cmu. edu/aura.*

[24] T. View, "Reconfigurable context-sensitive middleware for pervasive computing," *Pervasive Computing, IEEE*, vol. 1, no. 3, pp. 33–40, 2002.

[25] B. K. L. R. T. B. H. E. Maddy Janse, Fano Ramparany, "Specification of the amigo abstract system architecture," *IST Amigo Project Deliverable D2.3.*

[26] L. da Silva, P. Vink, and R. Poortinga-van Wijnen, "A Service-Oriented Middleware for Providing Context Awareness and Notification,"

[27] D. Wo, J. Tigli, S. Lavirotte, and M. Riveill, "Wcomp : a Multi-Design Approach for Prototyping Applications using Heterogeneous Resources," *17th IEEE International Workshop on Rapid System Prototyping (RSP)*, 2006.

[28] Donsez, "Atelier« mise en œuvre d'UPnP avec OSGi»,"

[29] H. C. D. C.-F.-W. P. C. G. D. V. H. S. L. S. M. A. O. A.-M. P.-D. N. R. L. S. e. J.-Y. T. Fabien Balligand, Mireille Blay-Fornarino, "Identification des modalités de prise en charge des contrats pour chaque plate-forme cible," tech. rep., RNTL Faros, 2007.

[30] S. Lavirotte, D. Lingrand, and J. Tigli, "Définition du contexte : fonctions de coût et méthodes de sélection," *Proceedings of the 2nd French-speaking conference on Mobility and uibquity computing*, pp. 9–12, 2005.

[31] C. T. Nabiha Belhanafi Behlouli, "Camido : un intergiciel pour la sensibilité au contexte," *Ubimob'06 Atelier d'étude du contexte*, Septembre 2006.

[32] H. Duran-Limon, G. Blair, A. Friday, P. Grace, G. Samartzidis, T. Sivaharan, and M. Wu, "Resource Management for the Real-Time Support of an Embedded Publish/Subscribe System," *Submitted to the 9th IEEE Real-Time/Embedded Technology and Applications Symposium (RTAS'03)*, 2003.

[33] D. Cheung, J. Tigli, S. Lavirotte, and M. Riveill, "Wcomp : a Multi-Design Approach for Prototyping Applications using Heterogeneous Resources," *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping, Chania-Crete*, 2006.

[34] C. Sørensen, M. Wu, T. Sivaharan, G. Blair, P. Okanda, A. Friday, and H. Duran-Limon, "A context-aware middleware for applications in mobile Ad Hoc environments," *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pp. 107–110, 2004.

[35] R. Douence, O. Motelet, and M. Sdholt, "A formal definition of crosscuts," *Metalevel Architectures and Separation of Crosscutting Concerns : Third International Conference, Reflection 2001, Kyoto, Japan, September 25-28, 2001 : Proceedings*, 2001.

[36] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. Young, "Four Easy Pieces for Assessing the Usability of Multimodal Interaction : The CARE Properties," *Proc. Of IFIP Int. Conf. on Human-Computer Interaction Interact*, vol. 95, pp. 115–120.

[37] J. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[38] F. Vernier and L. Nigay, "Interfaces multimodales : composition et caractérisation des modalités de sortie," *Actes de la conférence ErgoIHM*, pp. 203–210, 2000.

[39] V. Hourdin, S. Lavirotte, J. Tigli, and E. Rainbow, "Comparaison des systèmes de services pour dispositifs,"

[40] T. Gu, H. Pung, and D. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, 2005.

[41] S. Yau and F. Karim, "An Adaptive Middleware for Context-Sensitive Communications for Real-Time Applications in Ubiquitous Computing Environments," *Real-Time Systems*, vol. 26, no. 1, pp. 29–61, 2004.

[42] P. David and T. Ledoux, "Une approche par aspects pour le développement de composants Fractal adaptatifs," *RSTI-Série L'Objet (RSTI-Objet)*, vol. 12, no. 2-3, 2006.

Résumé

Software applications, now distributed or mobile, requires ever more abilities to support adaptation because of the multiplicity of contexts they have to deal with (multi-devices, physical environment high variability ...). Driven by the emergence of pervasive computing,many approaches have been proposed for building software that can dynamically adapt to their environment, so in order to do this they provide differents kind of mechanisms. The additional problem to be addressed is studying their triggering mechanisms, while maintaining an innovative architectural approach of decentralization particularly appropriate for ubiquitous computing.